UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Nara Sueina Teixeira

ANÁLISE DA COMPATIBILIDADE DE COMPONENTES ESPECIFICADOS EM UML

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Ricardo Pereira e Silva

Florianópolis
2012

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:

http://www.bu.ufsc.br/design/Catalogacao.html

Nara Sueina Teixeira

ANÁLISE DA COMPATIBILIDADE DE COMPONENTES ESPECIFICADOS EM UML

Esta Dissertação foi julgada adequada para obtenção do Título de Mestre, e aprovada em sua forma final pelo Programa de Pós Graduação em Ciência da Computação.

Florianópolis, 22 de março de 2012.

_____
Prof., Ronaldo dos Santos Mello, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof., Dr. Ricardo Pereira e Silva,
Orientador
Universidade Federal de Santa Catarina

_____
Prof., Dr. Jean Marie Farines,
Universidade Federal de Santa Catarina

_____
Prof., Dr. Marcello Thiry Comicholi da Costa,
Universidade do Vale do Itajaí

_____
Prof.ª, Dr.ª Patrícia Vilain,
Universidade Federal de Santa Catarina

Para minha mãe, Maria Vandi.

AGRADECIMENTOS

Agradeço a Deus por sempre atender aos meus pedidos e por toda luz enviada.

Ao meu orientador, professor Ricardo, pela paciência, desafios lançados e as longas conversas que me deixavam um pouco mais tranquila e me faziam acreditar que tudo daria certo.

Ao meu namorado, sempre companheiro, Leonardo, por me mostrar que o amor tudo sofre, tudo crê, tudo espera, tudo suporta.

Aos meus pais, em especial a minha mãe, Maria Vandi, pelo eterno incentivo aos estudos.

As minhas tias, Marias V, pelo amparo emocional, em especial à tia Valdênia, pelo exemplo de dedicação aos estudos e pelas palavras sempre tão sábias.

Aos meus colegas de laboratório Roberto Cunha, Leonardo Brasil e Ademir Coelho pela sempre disponibilidade em ajudar nos momentos de desespero para desvendar os mistérios do OCEAN.

À Universidade Federal de Santa Catarina, ao INE e a Katiana, pela eficiência em esclarecer prontamente todas as minhas dúvidas administrativas.

"Tudo posso naquele que me fortalece."

(Fl 4:13)

RESUMO

Na abordagem de desenvolvimento orientado a componentes, um aplicativo é produzido por meio da conexão de dois ou mais componentes. O aumento da previsibilidade do resultado da combinação de um componente com outros ainda é uma questão em aberto para pesquisa. A análise de compatibilidade de componentes é realizada a partir da descrição da interface dos componentes interligados. A maioria das propostas de análise de compatibilidade pesquisadas se restringe à verificação da ligação entre dois componentes e não considera todo o conjunto de componentes de uma aplicação. E quando o faz, limita-se a detectar apenas *deadlock* no sistema. Este trabalho propõe uma estratégia para a realização automatizada da análise comportamental de componentes a partir de uma abordagem descrita na literatura onde a descrição da interface de componentes é feita integralmente em UML. A descrição estrutural utiliza os diagramas de componentes, classe e implantação, e a comportamental, o diagrama de máquina de estados. A estratégia proposta estabelece uma forma de conversão da máquina de estados dos componentes individuais e da aplicação para redes de Petri, de forma transparente para o usuário, e define critérios para a análise comportamental. Problemas comportamentais são identificados a partir da interpretação das propriedades das redes de Petri no contexto dos componentes. Essa solução considera o comportamento do sistema como um todo e permite a identificação, além de *deadlock,* de possíveis problemas comportamentais, tratadas como advertências e que devem ser analisadas pelo usuário, tais como: serviços indisponíveis, temporariamente disponíveis ou serviços disponíveis no componente que passam a ser temporariamente disponíveis ou até indisiponíveis na aplicação. A solução proposta permite uma maior previsibilidade no resultado da combinação de componentes. Essa solução foi automatizada no ambiente SEA, por meio de ferramentas automatizadas que emitem relatórios com os problemas encontrados. São apresentados os estudos de caso realizados para avaliação da proposta.

Palavras-chave: Desenvolvimento orientado a componentes, análise de compatibilidade estrutural, análise de compatibilidade comportamental, UML, redes de Petri.

ABSTRACT

In the component-based software development, an application is produced by connecting two or more components. Increasing predictability of the result of combination of a component with others components is still an issue for research. The compatibility analysis of components is performed from the interface's description of the interconnected components. Most proposals for compatibility analysis are limited to verify the compatibility between only two components and does not consider the whole component set of an application. And when they do it, they only detect deadlock in the system. This study proposes a strategy to achieve automated behavioral analysis of components from an approach described in the literature, in which the interface of components is described entirely in UML.The structural description uses the component, class and deployment diagrams, and behavior description, the state machine diagram diagram. The proposed strategy provides a way of converting the state machine of the individual components and of the application in Petri nets, transparently to the user, and sets criteria for behavioral analysis. Behavioral problems are identified from the interpretation of the Petri nets properties in the context of the components. This solution considers the behavior of the system as a whole and allows the identification, in addition to deadlock, of potential behavioral problems, and treats then as warnings that must be analyzed by the user, such as: unavailable services, temporarily available services, or component available services become temporarily available or unavailable in the application. The proposed solution allows a more predictable result of the combination of components. This solution has been automated in the SEA environment, using automated tools included in it, where reports are issued with the found problems. The case studies produced to evaluate the proposal are presented.

Keywords: Component-based software development, structural compatibility analysis, behavioral compatibility analysis, UML, Petri net.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACM – Association for Computing Machinery
ADL – Architecture Description Language
CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
DSBC – Desenvolvimento de Software Baseado em Componentes
FAC – Ferramenta de Análise Comportamental
FAE – Ferramenta de Análise Estrutural
IC – Interface de Componentes
IEEE – Institute of Electrical and Electronic Engineers
OMG – Object Management Group
SOAP – Simple Object Access Protocol
UML – Unified Modeling Language
WSDL – Web Service Definition Language

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho se situa no contexto do desenvolvimento de software orientado a componentes e é voltado à análise de compatibilidade entre componentes. A partir de uma abordagem de descrição de software orientado a componentes existente usando exclusivamente UML, é proposta uma estratégia para a realização da análise de compatibilidade comportamental, utilizando redes de Petri. A proposta estabelece uma forma de converter parte do modelo UML em redes de Petri e define critérios para a análise comportamental. A solução proposta para a análise comportamental foi automatizada no ambiente SEA. Para avaliação da proposta foi necessária também a automatização da análise estrutural já descrita na literatura.

Este capítulo inicia com a contextualização do tema estudado. É apresentado o problema existente na área sobre o qual se desenvolveu a pesquisa, assim como os objetivos da pesquisa, o método escolhido para alcançá-los e as justificativas para o estudo. Por fim, são apresentados os resultados esperados com a conclusão do trabalho, suas limitações e sua estrutura.

1.1 CONTEXTUALIZAÇÃO

Um estudo realizado pela SOFTEX em parceria com o Departamento de Política Científica e Tecnológica da Unicamp, e com apoio do Ministério de Ciência e Tecnologia, indica a tendência cada vez maior das empresas de desenvolvimento de software de buscar junto à engenharia de software baseada em componentes o suporte às suas atividades. Este estudo indica alterações no mercado, com o estabelecimento de um novo segmento, vinculado à potencialização do reuso de software a partir da utilização de componentes (SOFTEX, 2007).

> Na década de 1990, a engenharia de software baseada em componentes ganhou projeção como uma abordagem para desenvolvimento de software com base no reuso de componentes. Ela consiste no processo de definir, implementar, integrar ou compor componentes independentes, pouco acoplados em sistemas (SOMMERVILLE, 2011, p.315).

"O desenvolvimento de software baseado em componentes (DSBC) é voltado ao tratamento da complexidade". (SILVA, 2009, p.173). A generalização da arquitetura cliente / servidor, a utilização de serviços distribuídos em diferentes nodos do sistema, a distância e o tempo real são alguns dos aspectos críticos cada vez mais envolvidos no desenvolvimento de software. Tais aspectos contribuem para a complexidade. Os clientes exigem softwares mais confiáveis, entregues e implantados mais rapidamente.

> O DSBC encarna as boas práticas da engenharia de software. Faz sentido projetar um sistema usando componentes, mesmo que seja necessário desenvolver, em vez de reusar esses componentes. (SOMMERVILLE, 2011, p.316).

Componentes desenvolvidos são passíveis de reuso futuro. O DSBC possui duas perspectivas:

1. **O desenvolvimento de componentes ou desenvolvimento para reuso**: processo interessado na criação de novos componentes, desde a especificação (análise, documentação) até a implementação. A documentação é o fator mais importante, pois é ela que possibilitará a reutilização desse componente em outros sistemas, caso haja a necessidade de alterações.
2. **O desenvolvimento com componentes ou desenvolvimento com reuso**: processo de desenvolvimento de novas aplicações usando um conjunto de componentes interligados, levando em conta que esses componentes já existam e que estejam disponíveis para uma seleção e reutilização. (SOMMERVILLE, 2011, p.321).

Os principais argumentos a favor da abordagem são a promoção do reuso de software e a intercambialidade (SOFTEX, 2007) (SILVA, 2009). Devido à separação de software em blocos de construção reutilizável, esse paradigma facilita o desenvolvimento, implementação e manutenção das aplicações. São essas "fatias" de softwares que chamamos de componentes. Um componente é constituído por uma parte externamente visível, a interface de componente, e uma não visível

externamente, a estrutura interna. A visão externa de um componente seria, portanto, a descrição de sua interface: isso inclui a estrutura da interface de componentes, restrições estabelecidas para a interação com outros componentes, além da descrição de suas funcionalidades (SILVA, 2009). As interfaces de componentes refletem os métodos que os componentes fornecem (interface fornecida) e os métodos de que o componente necessita para funcionar corretamente (interface requerida).

Segundo Sommerville (2011) os componentes são abstrações de nível mais alto do que objetos e possuem as seguintes características:

- Componentes são independentes. Sua implementação é ocultada e pode ser alterada sem afetar o restante do sistema;
- Componentes comunicam-se por meio de interfaces bem definidas;
- Componentes oferecem uma gama de serviços[1] que podem ser usados em sistemas de aplicações, o que reduz a quantidade de código novo a ser desenvolvido.

Assim, as principais motivações e oportunidades relacionadas ao DSBC, segundo a SOFTEX (2007), referem-se:

- Às vantagens técnicas para desenvolvimento de software baseado em componentes (qualidade, manutenibilidade, confiabilidade);
- Aos ganhos de produtividade, redução de prazo de entrega e de diversificação de produtos e soluções;
- À possibilidade de participação de pequenas e médias empresas no mercado internacional através da produção de componentes especializados para integradores e intermediários (*brokers)* que já tenham acesso a este mercado.

Além dos ganhos de produtividade associados ao reuso, encontram-se ganhos de qualidade e funcionalidade proporcionados pela tecnologia. Argumenta-se que o fato de um componente ser reutilizado em diferentes e numerosas situações faz com que ele seja mais testado e tenha seus erros corrigidos, atingindo a maturidade mais rapidamente. Outra vantagem técnica atribuída ao uso de componentes é que a tecnologia produz softwares mais flexíveis, duráveis e de manutenção

[1] Método e Serviço no contexto deste trabalho são considerados sinônimos.

facilitada, pois é possível, teoricamente, substituir um componente por um outro de melhor desempenho sem alterar o sistema (SOFTEX, 2007).

A tecnologia de componentes de software refere-se a todas as tecnologias relacionadas ao desenvolvimento e uso de componentes de software, ou seja, todas as ferramentas que auxiliam no projeto, construção, combinação, configuração e customização final dos componentes ou aplicações construídas a partir de componentes, bem como o ambiente para a execução dos componentes (*framework*). Assim, há duas principais atividades ligadas a componentes: o desenvolvimento e a produção dos componentes e seu uso para desenvolvimento e produção de programas (SOFTEX, 2007).

1.2 DEFINIÇÃO DO PROBLEMA

Apesar das vantagens do DSBC, a construção de software reutilizando componentes pré-fabricados ainda encontra algumas dificuldades que têm motivado o esforço de várias pesquisas na busca de soluções. Alguns dos problemas citados por Brereton (1999) e Szyperski (2003) são:

- Descrição de componentes: como descrever um componente para que possa ser facilmente encontrado e utilizado? Por mais simples que pareça, ainda é algo que não está plenamente estabelecido. Há necessidade de padronizações neste sentido, tanto para leitura e compreensão humana quanto na definição de interfaces entre componentes.
- Previsibilidade da composição: mesmo conhecendo todas as características de um componente, não há garantia quanto ao resultado da combinação do mesmo com outros componentes no desenvolvimento de um sistema. O aumento desta previsibilidade é uma tendência futura, mas ainda uma questão em aberto para pesquisa.
- Responsabilidade por falhas: a natureza do software faz com que seja difícil separar a fonte de uma determinada falha. Ainda há necessidade de criação de técnicas sólidas para identificar falhas geradas por partes do sistema (componentes) ou causadas pela integração.

- Suporte de ferramentas: a existência de ferramentas é essencial para o sucesso do DSBC (ferramentas de descrição e seleção de componentes, de avaliação, repositórios e testes).

Algumas pesquisas sugerem formas para descrever componentes e algumas propõem técnicas para a análise de compatibilidade (DIAS e VIEIRA, 2000), (SILVA, 2000), (PLASIL e VISNOVSKY, 2002), (CRAIG e ZUBEREK, 2004), (CUNHA, 2005), (CHOUALI e SOUQUIÈRES, 2005), (MOUAKHER, LANOIX e SOUQUIÈRES, 2006), (CRAIG e ZUBEREK, 2007) e (WEI e TONG, 2008).

Além desses problemas, questões relacionadas a componentes confiáveis, certificação de componentes, política de atualizações, riscos de mudança, modelos de negócio, suporte a longo prazo, propriedade intelectual e padrões dominantes também são discutidas em pesquisas (MANOLIOS, SUBRAMANIAN e VROON, 2007), (GROENDA, 2009) e (JUNG, 2011).

Apesar do esforço das pesquisas para solucionar os problemas relacionados ao DSBC, os problemas citados continuam a ser uma questão em aberto.

A especificação de um componente deve conter todas as informações necessárias para decidir se o componente pode ser usado em um dado contexto ou não. Os componentes devem ser completamente documentados para que os potenciais usuários possam decidir se satisfazem suas necessidades. A sintaxe e a semântica de todas as interfaces de componentes devem ser especificadas (SOMMERVILLE, 2011).

A partir da descrição da interface do componente deve ser possível analisar a compatibilidade desse componente com outros em uma aplicação baseada em componentes. Para garantir a compatibilidade dos componentes interligados, é necessário analisar a compatibilidade em todos os níveis: estrutural, comportamental e funcional. Por isso, a importância do uso de mecanismos de descrição de componentes capazes de descrevê-los considerando os três aspectos. A falta de um padrão para a descrição dos componentes dificulta o reuso dos mesmos, pois é a partir dessa descrição que é possível decidir sobre o uso, ou não, de um determinado componente.

Mais da metade das propostas de análise de compatibilidade pesquisadas neste trabalho se restringe à verificação da ligação entre dois componentes e não considera todo o conjunto de componentes de uma aplicação. E quando o faz, limita-se a detectar apenas *deadlock* no sistema.

A identificação de problemas que podem ocorrer no software baseado em componentes, decorrentes da má especificação dos componentes utilizados ou da conexão entre eles, é necessária para uma maior confiabilidade e qualidade do produto desenvolvido.

Devido à alta complexidade dos sistemas desenvolvidos atualmente, a identificação manual desses possíveis problemas pode ser uma tarefa exaustiva para os projetistas de software. O suporte de ferramentas para auxiliar na especificação e análise de compatibilidade entre componentes é fundamental para o avanço e prática da abordagem.

> Em tudo aquilo que disser respeito a vantagens produtivas técnicas e econômicas (custos, escala, tempo, qualidade, facilidade operacional, etc.) é de se esperar que o mercado de componentes se desenvolva. Entretanto, se não se formar um regime minimamente estruturado (estabelecimento de padrões, regime de propriedade, etc.) que garanta a remuneração do esforço de desenvolvimento e de venda de componentes, este mercado deve seguir fragmentado e pautado no desenvolvimento atomizado ou por comunidades de prática e arranjos similares (SOFTEX, 2007, p.7).

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Este estudo tem como principal objetivo a criação de mecanismos que possibilitem, de forma automática, uma maior previsibilidade de problemas relacionados à combinação de componentes em um software orientado a componentes, especificado em UML, dentro do ambiente SEA[2] ou de outros suportes ferramentais.

[2] SEA, detalhado no capítulo 2, é um ambiente de desenvolvimento de software em que o paradigma de orientação a objetos é usado para produção e uso de artefatos de software reutilizáveis.

1.3.2 Objetivos Específicos

A partir desse objetivo geral, este trabalho tem como objetivos específicos:

- Identificar possíveis problemas relacionados à compatibilidade de componentes.
- Definir uma estratégia de conversão das máquinas de estados para redes de Petri para realizar a análise de compatibilidade de componentes a partir da especificação feita integralmente com diagramas UML sugerida por Silva (2009).

- Estabelecer critérios de análise de compatibilidade comportamental a a partir das propriedades das redes de Petri.
- Relacionar como os problemas comportamentais devem ser identificados e exibidos aos usuários.
- Prover um mecanismo automatizado no ambiente SEA para avaliar a estratégia definida.

1.4 HIPÓTESE DE PESQUISA

A hipótese de pesquisa deste trabalho é:

"*A conversão da especificação UML do software orientado a componentes para redes de Petri pode fornecer o formalismo necessário para permitir a análise de compatibilidade entre os componentes interligados. A partir desta análise deve ser possível a identificação de problemas que podem ocorrer no software quando um componente é conectado a outros"*.

1.5 JUSTIFICATIVA

Uma questão fundamental relacionada à abordagem do DSBC diz respeito à compatibilidade de componentes, que deve ser analisada a partir da descrição dos componentes interligados.

Para alguns autores (RENAUX e LEFEBVRE, 2006), (BRUEL e OBER, 2006), (KONG et al, 2009), a UML tem se tornado um padrão para a modelagem de sistemas de software. Acredita-se que ela é provida de características tais como simplicidade, facilidade de uso devido a um certo grau de intuição e flexibilidade em sua notação.

Assim, as notações intuitivas dos diagramas UML facilitam a distribuição e comunicação de artefatos de software entre os envolvidos no seu desenvolvimento.

A UML fornece mecanismos para tratar componentes e a partir de sua versão 2.0 introduziu alguns novos conceitos (porto, conectores, etc) cruciais para descrever a arquitetura de sistema baseado em componentes (BRUEL e OBER, 2006) (SILVA, 2009). Porém, ela não estabelece um padrão para especificações de software.

Silva (2009) propôs uma abordagem estabelecendo convenções para especificar software baseado em componentes, no paradigma de orientação a objetos, exclusivamente com diagramas UML.

No entanto, a UML carece de formalismos em sua aplicabilidade para sistemas críticos, onde projetos rigorosos e precisos são altamente importantes para o desenvolvimento correto da aplicação. Isso dificulta, mas não impossibilita as análises e verificações automáticas. Abordagens que utilizam UML e que precisam realizar validações automáticas, relacionadas a um problema específico, independente do contexto, necessitam atribuir uma semântica formal ao modelo para permitir tais validações.

Para superar essa deficiência, uma das soluções é utilizar as vantagens das linguagens formais para dar uma precisão semântica aos diagramas da UML, por meio de transformações formais. Algumas linguagens formais utilizadas são: método B (MEYER e SOUQUIÈRES, 1999), linguagem Maude (MOKHATI, GAGNON e BADRI, 2007), linguagem PROMELA (LILIUS e PALTOR, 1999), (SCHÄFER, KNAPP e MERZ, 2001) e redes de Petri (KING e POOLEY, 1999), (SALDHANA e SHATZ, 2000), (BARESI e PEZZÈ, 2001), (THIERRY-MIEG e HILLAH, 2008) e (KONG et al, 2009).

A vantagem desse tipo de abordagem é a junção das boas características de cada uma das linguagens. As linguagens formais utilizam formalismos matemáticos bem fundamentados tais como teoria dos conjuntos, lógica de predicados, lógica modal, autômatos finitos, etc. Elas permitem determinar propriedades da especificação do software tais como consistência, completude, entre outras. Porém, esse tipo de linguagem não é amplamente utilizado na comunidade de engenharia de software, pois apesar de poder ser analisada por ferramentas específicas, que determinam propriedades importantes da especificação, possuem algumas limitações, como um baixo grau de abstração.

Cinco propostas dentre as pesquisadas neste trabalho utilizam redes de Petri para especificar aspectos relacionados a componentes e

realizar análises de compatibilidade (SILVA, 2000), (CRAIG e ZUBEREK, 2004), (CUNHA, 2005), (CRAIG e ZUBEREK, 2007) e (WEI e TONG, 2008). Porém, as análises realizadas se limitam a verificar apenas a propriedade de bloqueio na rede (*deadlock*). Além disso, as redes de Petri são utilizadas em conjunto com outros modelos, o que obriga os projetistas a utilizarem mais de uma linguagem para a especificação do sistema baseado em componentes.

Um modelo de descrição de componentes adequado deve possuir características como facilidade de compreensão e utilização para os projetistas de sistemas. Características dessa natureza são difíceis de serem medidas, por se tratarem de conceitos um tanto abstratos. Ao mesmo tempo, deve ser provido de um certo grau de formalismo para permitir que incompatibilidades sejam identificadas.

Em uma aplicação baseada em componentes, é necessário prever a maior quantidade possível de problemas relacionados à interligação de componentes. Redes de Petri possuem várias propriedades que podem ser analisadas considerando o contexto de componentes, na tentativa de encontrar problemas relacionados à compatibilidade entre eles. Considerando sua adequação e o fato de ter sido usada em esforços de pesquisa anteriores, neste trabalho optou-se pelo uso das Redes de Petri em conjunto com UML.

1.6 MÉTODO DE PESQUISA

Com base nas formas clássicas de classificação de pesquisas (SILVA; MENEZES, 2001) esse trabalho é classificado da seguinte forma:

- Natureza da pesquisa: *Aplicada*. A pesquisa aplicada objetiva gerar conhecimentos para aplicação prática, dirigidos à solução de problemas específicos. Envolve verdades e interesses locais. No presente caso, objetiva-se gerar conhecimento para aplicação prática na área de desenvolvimento de software orientado a componentes. A proposta deste trabalho visa auxiliar na solução do problema específico da previsão da compatibilidade entre componentes.
- Abordagem do problema de pesquisa: *Qualitativa.* A pesquisa qualitativa não requer o uso de métodos e técnicas estatísticas. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento-chave. Os pesquisadores tendem a

analisar seus dados indutivamente. O processo e seu significado são os focos principais de abordagem. Neste trabalho, buscou-se profunda compreensão sobre o desenvolvimento de software orientado a componentes e os problemas relacionados à compatibilidade de componentes interligados em uma aplicação por meio de revisão bibliográfica. Procurou-se, então, uma solução que permitisse identificar uma quantidade maior de problemas comportamentais do que os já identifcados nas pesquisas realizadas.

- Objetivos de pesquisa: *Pesquisa exploratória.* A pesquisa exploratória visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses. Envolve levantamento bibliográfico; entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; análise de exemplos que estimulem a compreensão. Assume, em geral, as formas de Pesquisas Bibliográficas e Estudos de Caso. Buscou-se conhecer o tema compatibilidade de componentes através de revisão bibliográfica sistemática e análise de vários exemplos de sistemas baseado em componentes para estimular a compreensão. Estudos de caso foram aplicados e serão vistos no capítulo 5.
- Procedimentos técnicos de pesquisa: *Pesquisa bibliográfica* e *Estudo de caso.* A pesquisa é bibliográfica quando elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e atualmente com material disponibilizado na Internet. O estudo de caso ocorre quando envolve o estudo profundo e exaustivo de um ou poucos objetos de maneira que se permita o seu amplo e detalhado conhecimento. Neste trabalho, a revisão de literatura foi elaborada por meio da metodologia de pesquisa bibliográfica baseada na identificação, coleta e análise de fontes literárias específicas, conforme será visto no capítulo 3. Estudos de caso foram realizados e serão vistos no capítulo 5.

A seguir, são pontuados os passos seguidos para realizar o trabalho:

- A partir da análise dos trabalhos correlatos, relacionar as formas usuais de especificação da interface de componentes, assim como técnicas para realizar a análise estrutural e comportamental de componentes. Além disso, identificar quais

problemas, relacionados à compatibilidade estrutural e comportamental, são verificados.

- A partir da abordagem sugerida por Silva (2009) para a especificação de componentes e de software baseado em componentes, utilizando apenas diagramas da UML:

    - implementar, no ambiente SEA, melhorias relacionadas ao suporte gráfico dos diagramas UML necessários para o desenvolvimento da abordagem proposta;
    - implementar, no ambiente SEA, uma nova ferramenta para viabilizar a geração automática da especificação comportamental da aplicação, representada pelo diagrama de máquina de estados;

- Definir um algoritmo de conversão do diagrama de máquina de estados da UML para redes de Petri, considerando a semântica estabelecida na abordagem, provendo o formalismo necessário para permitir a identificação automática de problemas relacionados à compatibilidade comportamental entre componentes por meio da interpretação das propriedades redes de Petri.

- Definir quais problemas, relacionados à compatibilidade comportamental de componentes, podem ser identificados a partir da interpretação das propriedades das redes de Petri. Associar a cada propriedade o problema relacionado ao contexto dos componentes.

- Desenvolver protótipos de relatórios que devem ser emitidos na análise de compatibilidade estrutural e comportamental de componentes, informando erros e possíveis problemas, para serem analisados pelos usuários da ferramenta.

- Implementar uma nova ferramenta para a análise estrutural de componentes no ambiente SEA. Essa ferramenta abrange:

    - a análise dos portos dos componentes conectados;
    - a emissão do relatório de análise estrutural listando os erros encontrados.

- Implementar uma nova ferramenta para a análise comportamental de componentes no ambiente SEA. Essa ferramenta abrange:

    - a implementação do algoritmo de conversão da máquina de estados da UML em redes de Petri;
    - a identificação das propriedades das redes de Petri;
    - a emissão do relatório de análise comportamental listando os erros e os possíveis problemas encontrados por meio da interpretação automática das propriedades das redes de Petri, no contexto dos componentes.

- Avaliar as ferramentas implementadas no ambiente SEA por meio de estudos de casos incluindo propositalmente incompatibilidades entre os componentes desenvolvidos.

- Analisar o desempenho da ferramenta, comparando seus resultados com os resultados verificados manualmente, para aferir o tempo de análise gasto em cada método (automático e manual).

1.7 RESULTADOS ESPERADOS

Com a conclusão desse trabalho, tem-se como resultados esperados:

- Previsão mais acurada da composição de componentes, a partir da definição de uma estratégica para análise de compatibilidade entre componentes de um software orientado a componentes especificados exclusivamente com diagramas da UML, identificando além de erros de compatibilidade, situações que merecem atenção e devem ser analisadas pelos usuários pela possibilidade de indicar erros no software.

- Contribuição para a padronização do uso de componentes com a adoção da abordagem incluindo a especificação sugerida por Silva (2009) e a estratégia definida nesse trabalho para a identificação de problemas de compatibilidade dentro de outras ferramentas que utilizam UML, além do ambiente SEA.

- Colaboração com as discussões sobre a especificação de software orientado a componentes e análise de compatibilidade entre componentes.

Espera-se que esses resultados contribuam para a melhoria da produtividade e qualidade de especificações de software orientado a componentes e, por conseguinte, para que as falhas nesse tipo de software sejam minimizadas.

1.8 LIMITAÇÕES DO TRABALHO

Este trabalho está focado na especificação de novos componentes de software e na verificação de compatibilidade entre eles em uma aplicação baseada em componentes, no paradigma orientado a objetos.

Não fará parte do escopo do trabalho a busca e seleção de componentes já existentes, o tratamento de adaptadores de componentes, assim como a implementação dos componentes, a partir da especificação em alguma linguagem ou modelo de componente específico. Essas e outras questões são perspectivas de trabalhos futuros, explicadas no capítulo conclusivo.

1.9 ESTRUTURA DO TRABALHO

Os demais capítulos desta dissertação estão estruturados como descrito a seguir:

Capítulo 2 – Componentes, UML, Redes de Petri e OCEAN/SEA: aborda os principais conceitos ao tema da dissertação e técnicas utilizadas na abordagem proposta. Serão explanados nas seguintes seções:

- Seção 2.1 – Desenvolvimento de software orientado a componentes.
- Seção 2.2 – UML
- Seção 2.3 – Redes de Petri
- Seção 2.4 – OCEAN/SEA

Capítulo 3 – Comparação com Trabalhos Correlatos: aborda os trabalhos correlatos a essa proposta e apresenta uma comparação entre as contribuições dessa pesquisa e as desses trabalhos.

Capítulo 4 – Apresentação da Abordagem para Especificação e Análise de Compatibilidade de Componentes: este capítulo apresenta a proposta sugerida nessa dissertação para a análise de compatibilidade entre os componentes a partir da especificação dos componentes, e do software baseado em componentes, feita em UML.

Capítulo 5 – Estudo de Caso: nesse capítulo são apresentados alguns estudos de caso utilizados para avaliar a proposta e as ferramentas desenvolvidas.

Capítulo 6 – Conclusões, Contribuições e Perspectivas Futuras: apresenta as considerações finais sobre o trabalho, conclusões alcançadas, contribuições e sugestões de continuidade da pesquisa.

2 COMPONENTES, UML, REDES DE PETRI E OCEAN/SEA

Neste capítulo são apresentados conceitos e técnicas utilizadas nesse trabalho, assim como o ambiente de desenvolvimento de software utilizado para desenvolver, testar e avaliar a proposta apresentada.

Inicialmente são apresentados conceitos referentes ao desenvolvimento de software orientado a componentes. Em seguida, é dada uma visão geral da linguagem de modelagem unificada, UML, e dos diagramas utilizados na abordagem sugerida. O capítulo segue com a apresentação das redes de Petri, modelo formal utilizado na análise comportamental de componentes. E, por fim, é dado um breve histórico do *framework* OCEAN e do ambiente SEA, no qual ferramentas foram implementadas e inseridas.

2.1 DESENVOLVIMENTO DE SOFTWARE ORIENTADO A COMPONENTES

> O desenvolvimento orientado a componentes é um paradigma de desenvolvimento em que um artefato de software é construído como a composição de uma coleção de unidades construtivas: componentes. É voltado ao tratamento da alta complexidade, pois a organização em componentes é basicamente uma estratégia de quebra de complexidade. Além disso, o outro argumento a favor da abordagem é a promoção do reuso de software: na medida em que mais componentes tornam-se disponíveis, podem ser inseridos em bibliotecas de componentes. Com a maior disponibilidade, a expectativa é a de que parte dos componentes dos desenvolvimentos esteja previamente pronta e seja reusada. (SILVA, 2009, p.173).

2.1.1 Componentes

Apesar da ideia inicial do desenvolvimento de software orientado a componentes ter mais de 40 anos, uma das primeiras definições de componentes encontradas na literatura apareceu somente em 1996, no evento Workshop on Component-Oriented Programming (WCOP):

“uma unidade de composição com interfaces contratualmente especificadas e dependência de contexto explícita. Componentes podem ser duplicados e estar sujeitos à composição com terceiros” (SZYPERSKI et al., 1996).

No evento do ano seguinte, o WCOP 97, essa visão foi refinada: “o que torna alguma coisa um componente não é uma aplicação específica nem uma tecnologia de implementação específica. Assim, qualquer dispositivo de software pode ser considerado um componente, desde que possua uma interface definida. Esta interface deve ser uma coleção de pontos de acesso a serviços, cada um com uma semântica estabelecida” (SZYPERSKI et al., 1997).

Atualmente, ainda não existe um consenso sobre a definição de componente de software (SOMMERVILLE, 2011). O termo componente é utilizado de uma forma bastante genérica sendo empregado para descrever diferentes conceitos, inclusive modelos de componentes, que são dependentes de uma determinada tecnologia, como Javabeans™ da Sun™ e Component Object Model (COM™) e sua extensão, COM™ distribuído (DCOM™), da Microsoft™.

Councill e Heineman (2001) estabelecem uma relação entre modelo e componente em sua definição: “um elemento de software que está de acordo com um modelo de componente padrão e pode ser independentemente implantado e composto, sem modificações, de acordo com um padrão de composição”. Nessa definição, entende-se que para os autores os componentes são como uma parte de código implementado em um modelo padão que pode ser injetada no código de qualquer sistema que queira utilizá-lo.

Brown (1996) define componente como “uma parte não-trivial, quase independente, e substituível de um sistema que cumpre uma função clara no contexto de uma arquitetura bem definida”. Mesmo não definindo uma relação direta entre componente e modelo, o autor deixa dúvidas quando menciona a característica “quase independente”. Essa característica pode ser interpretada como algo específico que torna o componente dependente de alguma forma de uma tecnologia.

Já Szyperski (2002), assim como Brown, não menciona padrões em sua definição, mas, em vez disso, concentra-se nas principais características dos componentes: “um componente de software é uma unidade de composição com interfaces contratualmente especificadas e apenas dependências de contexto explícitas. Um componente de software pode ser implantado de forma independente e está sujeito a ser composto por parte de terceiros”. Essa definição é mais genérica do que as citadas anteriormente.

Segundo Johnson (2011) componentes não caracterizam uma tecnologia. O autor define de forma explícita a separação entre componentes e tecnologia.

Falkowski (2010) define componente como uma unidade de software reusável, de acordo com um modelo de componente específico, com uma interface bem definida, encapsulando sua realização e possíveis estados internos.

Pela definição desses autores é possível perceber que de fato ainda não existe uma definição padrão para componentes. A principal diferente entre as definições citadas acima é a presença ou não de um determinado modelo padrão e tecnologia específica. Esse é um ponto importante nesta dissertação já que o termo componente é utilizado conforme a definição do OMG (2011): "Um componente representa uma parte modular de um sistema que encapsula seu conteúdo e cuja manifestação é substituível dentro de seu ambiente".

A abordagem apresentada considera os aspectos da especificação de componentes, que são interligados para a composição de um software orientado a componentes, abstraindo detalhes de implementação, isto é, não trata da implementação de componentes em uma determinada tecnologia e não utiliza nenhum modelo de componente específico.

Algumas características básicas de componentes, segundo a SOFTEX (2007) são:

- componentes são intercambiáveis;
- componentes são reutilizáveis;
- alguns são de uso mais geral e outros têm uso mais específico;
- componentes interagem com outros componentes.

O aumento da atenção ao desenvolvimento orientado a componentes, ocorrido a partir da década de noventa, segundo Seetharaman (1988 apud SILVA, 2009), se deve, porém, à disponibilidade de mecanismos de interconexão, como Corba, que permite a operação conjunta de componentes, independente de sua linguagem de programação, plataforma de execução e localização física.

Os mecanismos de conexão de componentes não fazem parte do foco desde trabalho – portanto, não é dada ênfase a este assunto. Para maiores informações sugere-se utilizar como referência (SZYPERSKI, 2002), que aborda o tema de forma detalhada.

2.1.1.1 Componentes e serviços

Uma das soluções mais utilizadas nos dias atuais para a comunicação entre aplicações de diferentes plataformas são os *web services*. Essa solução permite às aplicações enviar e receber dados em formato XML (*Extensible Markup Language*). Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML.

Geralmente, um *web service* é uma representação padrão para algum recurso computacional ou de informações que pode ser usado por outros programas, os quais podem ser recursos de informações, como um catálogo de peças, recursos de computador, tais como um processador especializado, ou recursos de armazenamento. A essência de um serviço é que o fornecimento do serviço é independente da aplicação que o usa. Os provedores de serviços podem desenvolver serviços especializados e oferecê-los para uma variedade de usuários de diferentes organizações (SOMMERVILLE, 2011).

Web services podem implementar uma arquitetura orientada a serviços (SOA, *Service-Oriented Architecture*). SOA é uma arquitetura de software onde as funcionalidades implementadas pelas aplicações são disponibilizadas na forma de serviços.

Os provedores de serviço projetam e implementam serviços e especificam sua interface. Eles também publicam informações sobre esses serviços em um registro acessível. Os solicitantes de serviços descobrem a especificação do serviço a ser utilizado e localizam seu provedor. Em seguida, eles podem ligar sua aplicação a esse serviço específico e comunicar-se com ele, usando protocolo de serviço padrão.

Em resumo, os principais padrões para SOA na Web são:

> - SOAP (*Simple Object Access Protocol*): padrão de troca de mensagens que oferece suporte à comunicação entre os serviços.
> - WSDL (Web *Service Definition Language*): a linguagem de definição de *web service* é um padrão para a definição de interface de serviço. Define como as operações (nomes de operações, parâmetros e seus tipos) e associações de serviço devem ser definidas.
> - WS-BPEL: Padrão para uma linguagem de *workflow*, que é usada para definir programas de processos que envolvem vários serviços diferentes. (SOMMERVILLE, 2011, p.357)

Segundo Sommerville (2011), uma distinção fundamental entre um serviço e um componente de software é que os serviços são projetados para serem independentes e podem ser usados em contextos diferentes. Portanto eles não têm interfaces requeridas, que no DSBC, define os serviços dos outros componentes que devem estar presentes no sistema. Ao contrário dos componentes de software, os serviços não usam chamadas de procedimentos ou de métodos remotos para acessar a funcionalidade associada a outros serviços.

2.1.2 Interface de Componentes

A interface de componentes (IC) é "uma coleção de pontos de acesso a serviços, cada um com semântica estabelecida". (SZYPERSKI et al., 1997).

Ela estabelece os serviços fornecidos e requeridos por um componente, sem considerar detalhes de implementação.

A descrição da interface de um componente abrange três aspectos distintos: estrutural, comportamental e funcional. A descrição estrutural refere-se aos aspectos estáticos de componentes e corresponde à relação das assinaturas dos métodos requeridos e fornecidos da IC. A descrição comportamental especifica restrições na ordem de invocação de métodos requeridos e fornecidos. Esses aspectos se atêm à interface do componente. A descrição funcional vai além da interface de componentes e descreve o que o componente faz – sem necessariamente entrar no detalhe de sua implementação (SILVA, 2000).

A descrição, nesse contexto, refere-se à especificação da IC, que possui um papel fundamental da abordagem do desenvolvimento de software orientado a componentes, pois possibilita a análise de compatibilidade entre os componentes interligados em uma aplicação. No entanto, ainda não existe um padrão amplamente aceito para a especificação da IC, o que dificulta a análise de compatibilidade e, consequentemente, o reuso dos componentes. Algumas abordagens são discutidas no capítulo 3.

2.1.3 Compatibilidade de Componentes

Para que diferentes componentes desenvolvidos independentemente possam trabalhar juntos, eles devem interagir de forma

satisfatória, isto é, suas interfaces devem ser compatíveis através de diferentes níveis, evitando assim erros de comunicação e problemas no sistema desenvolvido. Os níveis de compatibilidade seguem a mesma classificação da descrição da IC: estrutural, comportamental e funcional.

Dois componentes são estruturalmente compatíveis se o conjunto de métodos requeridos através da interface de um está disponível na interface do outro. Eles são compatíveis no nível comportamental quando as restrições associadas à ordem de execução de métodos, fornecidos ou requeridos, estabelecidas em um são respeitadas pelo outro. E são funcionalmente compatíveis quando as funcionalidades requeridas por um são supridas pelo outro (SILVA, 2000).

A verificação da compatibilidade entre componentes é realizada a partir da especificação da IC, portanto, essa descrição deve conter informações suficientes para possibilitar essa análise. Por isso a importância do uso de mecanismos de descrição de componentes capazes de descrevê-los estrutural, comportamental e funcionalmente. Para garantir que os componentes interligados em uma aplicação são compatíveis, é necessário analisar a compatibilidade nos três níveis: estrutural, comportamental e funcional.

Esse trabalho não visa assegurar a compatibilidade dos componentes interligados em uma aplicação, e sim a identificação de possíveis incompatibilidades nos níveis estrutural e comportamental. O nível funcional não faz parte do escopo deste trabalho, podendo ser explorado em trabalhos futuros.

A identificação de incompatibilidades, sejam elas estruturais ou comportamentais, são necessárias para decidir sobre o uso ou não de um componente no sistema. Um dos desafios das pesquisas é tentar prever a maior quantidade possível de problemas que podem ocorrer na conexão de componentes. Por exemplo, a conexão de dois componentes, caso eles não sejam compatíveis, pode bloquear a execução de um sistema. Essa situação deve ser evitada no desenvolvimento de um sistema, para que o mesmo não seja concebido com erros. Existem situaçãoes que podem representar problema para aplicação ou não, vai depender da análise do usuário. Essas e outras questões sobre compatibilidade estrutural e comportamental de componentes são discutidas em detalhes no capítulo 4.

A adaptação de componentes está diretamente relacionada com a identificação de incompatibilidade entre eles. Segundo Bosch (1997 apud SARTORI, 2005) componentes encontrados prontos, dificilmente podem ser usados e combinados uns com os outros, dentro de uma aplicação, exatamente “como são”, pois é muito difícil prever todas as

aplicações possíveis e todos os ambientes e circunstâncias em que ele pode ser usado.

Esse trabalho não aborda o tema adaptação de componentes, que também pode ser explorado em trabalhos futuros.

2.2 UML

Segundo o OMG (*Object Management Group*), entidade internacional que define e ratifica padrões na área de orientação a objetos, a UML *(Unified Modeling Language*, ou "Linguagem de Modelagem Unificada") é uma linguagem visual para especificação, construção e documentação de artefatos de software.

Nesta seção, é apresentada uma visão geral da UML e diagramas utilizados neste trabalho.

2.2.1 Visão Geral

A UML teve origem em 1994, a partir da compilação das melhores práticas de engenharia de software da época, que provaram ter sucesso na modelagem de sistemas grandes e complexos. Ela é resultado da fusão das notações Booch (de *Grady Booch*), OMT (de *James Rumbaugh*) e OOSE (de *Ivar Jacobson*). Em 1996 foi lançada sua primeira versão e a partir de 1997 foi aprovada como padrão internacional para modelagem de software orientado a objetos pelo OMG.

A UML não é uma metodologia de desenvolvimento, isto é, ela não estabelece uma série de passos ou atividades para obtenção de um produto de software. Ela auxilia a modelagem de software, que normalmente implica na construção de modelos gráficos que simbolizam os artefatos de software utilizados e seus inter-relacionamentos.

A UML provê benefícios tais como: visualização do desenho do sistema; gerenciamento de complexidade e clareza de comunicação. Tais benefícios permitem a identificação e o tratamento de problemas em etapas iniciais do processo de desenvolvimento do software, isto é, quando ele está sendo concebido.

Até a versão 1.5 (OMG, 2003), publicada em 2003, a UML possuía limitações em representar conceitos de software orientado a componentes, o que foi resolvido a partir da versão 2.0 (OMG, 2005), publicada em 2005.

A versão 2.0 (OMG, 2005), incorporou diagramas e elementos que representam de forma precisa conceitos de componentes e de software orientado a componentes, com extensões ao diagrama de componentes e a inclusão do diagrama de estrutura composta (SILVA, 2009).

Um diagrama é uma representação gráfica de um conjunto de elementos (classes, interfaces, colaborações, componentes, nós, etc) utilizados para visualizar o sistema modelado sob diferentes perspectivas. A UML define um número de diagramas que permite dirigir o foco para aspectos diferentes do sistema de maneira independente.

Atualmente, a UML é composta por catorze diagramas, classificados em diagramas estruturais e de comportamento. A figura 2.1 apresenta a estrutura das categorias dos diagramas da UML (OMG, 2011).

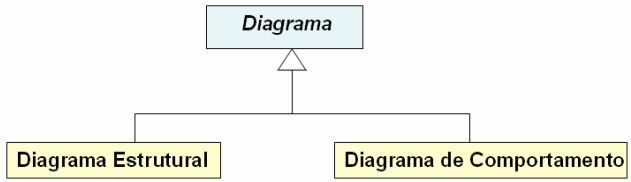


Figura 2.1: Categoria de diagrama da UML (OMG, 2011).

Os diagramas estruturais, ilustrados na figura 2.2, conforme a especificação da OMG (OMG, 2011), tratam o aspecto estrutural do sistema, isto é, o conjunto de elementos que compõem um software orientado a objetos e seus relacionamentos. Os aspectos estáticos de um sistema de software abrangem a existência de itens como classes, interfaces, colaborações e componentes.

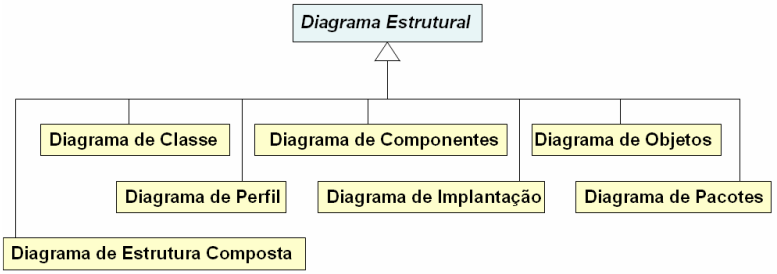


Figura 2.2: Diagramas estruturais da UML (OMG, 2011).

Os diagramas de comportamento, ilustrados na figura 2.3, conforme a especificação da OMG (OMG, 2011), são voltados a descrever o sistema computacional modelado quando em execução, representando a modelagem dinâmica do sistema. Os aspectos dinâmicos de um software compreendem as partes que "sofrem alterações", como por exemplo, o fluxo de mensagens ao longo do tempo.

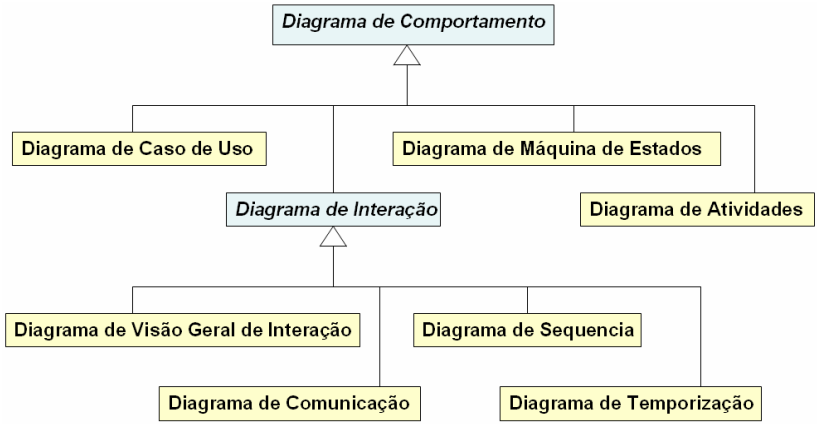


Figura 2.3: Diagramas de comportamento da UML (OMG, 2011).

A seguir são apresentados sumariamente os diagramas da UML utilizados nesse trabalho, com exemplos de utilização meramente ilustrativos.

2.2.2 Diagrama de Componentes

O diagrama de componentes descreve os componentes de software e seus relacionamentos. Seus principais elementos sintáticos são:

- Componente: elemento lógico que representa uma parte modular de um sistema.
- Interface (UML): uma relação de assinaturas de métodos que correspondem aos métodos fornecidos ou requeridos de um componente ao qual a interface esteja associada.

- Porto: “os portos formalizam os pontos de interação em um componente, e oferecem o mapeamento explícito entre interfaces fornecidas e requeridas”. (PENDER, 2004, p.425). A um porto podem ser associadas uma ou mais interfaces (UML) que especificam os métodos acessíveis naquele ponto de conexão. (SILVA, 2007).

É importante destacar a diferença de IC e interface (UML). A IC se refere à parte de um componente responsável pela sua comunicação com o meio externo e interface (UML) se refere à relação de assinaturas de métodos, que, no caso de um componente, é associada a um ou mais de seus portos (SILVA, 2009).

Em um diagrama de componentes é possível ter os seguintes relacionamentos:

- Conectores de montagem;
- Conector de delegação;
- Realização;
- Dependência.

As relações de dependência e realização podem ser inseridas no diagrama de componentes para interligar portos às interfaces (UML), representando os serviços requeridos e fornecidos pelos componentes.

A figura 2.4 ilustra um exemplo de diagrama de componentes.

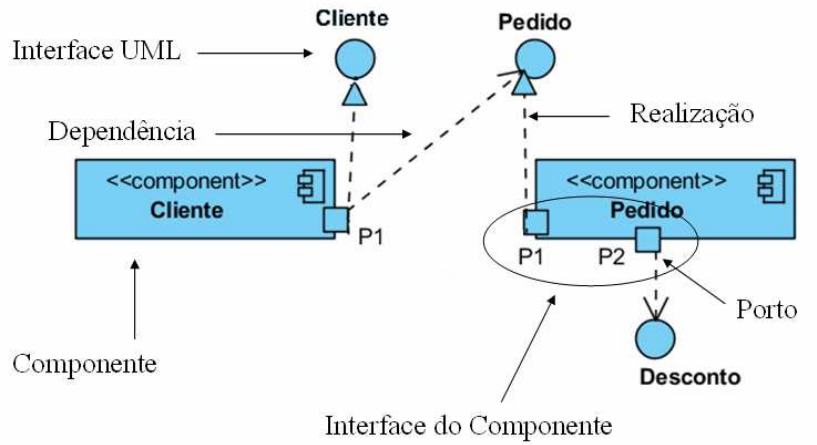


Figura 2.4: Diagrama de componentes.

Nesse trabalho, o diagrama de componentes é utilizado para declarar os componentes que serão interligados em um software orientado a componentes.

2.2.3 Diagrama de Classes

O diagrama de classes é o modelo fundamental de uma especificação orientada a objetos. Produz a descrição mais próxima da estrutura do código de um programa, isto é, mostra o conjunto de classes com seus atributos e métodos e os relacionamentos entre classes. Classes e relacionamentos constituem os elementos sintáticos básicos do diagrama de classes (SILVA, 2007).

Os relacionamentos possíveis entre classes são:

- Herança;
- Agregação;
- Composição;
- Associação;
- Realização;
- Dependência.

A figura 2.5 ilustra um exemplo do diagrama de classes.

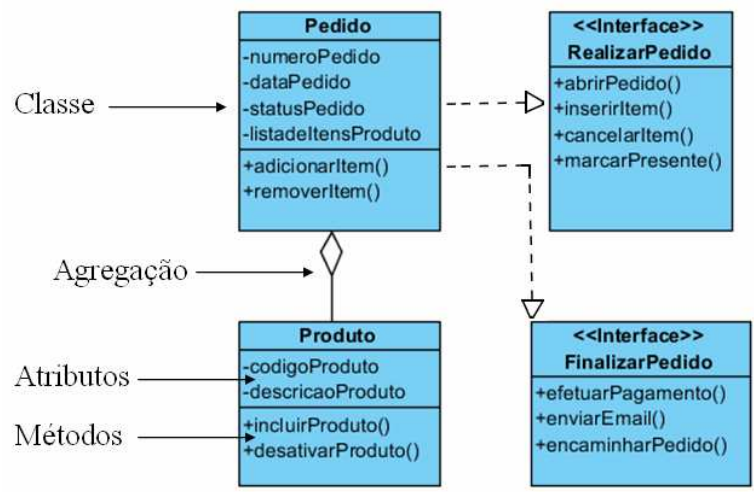


Figura 2.5: Diagrama de classes.

Nesse trabalho, o diagrama de classes é utilizado para declarar as interfaces associadas aos portos dos componentes declarados no diagrama de componentes.

2.2.4 Diagrama de Máquina de Estados

> A origem do diagrama de máquina de estados de UML é o *statechart* de Harel (1987). O modelo de Harel apresenta vantagens em relação a outros modelos voltados a descrever máquina de estados, como a possibilidade de descrever paralelismo e detalhamento de estados em subestados. O diagrama de máquina de estados da segunda versão da UML preserva essas características e adiciona elementos que permitem estabelecer referências entre diferentes diagramas, o que é particularmente útil para suportar modelagem de sistemas de mais alta complexidade (SILVA, 2007, p.142).

A modelagem dinâmica através da máquina de estados consiste em identificar o conjunto de situações em que o elemento modelado pode estar envolvido ao longo do período tratado pela modelagem e os percursos possíveis desse conjunto de situações. Estado é o nome dado a cada situação e transição, à passagem de um estado para outro (SILVA, 2007).

O diagrama de máquina de estados tem como elementos principais o estado e a transição, descritos da seguinte forma por SILVA (2007):

- estado: um estado pode corresponder a uma situação na qual o sistema permanece até que um evento o faça sair dela, ou à execução de uma tarefa – sendo que o final da execução acarretaria a saída da situação. A representação do estado é um retângulo de cantos arredondados, com o nome do estado dentro. Divisões podem existir para conter outras informações associadas a um estado.
- transição: é um elemento que produz a pasagem de um estado (ou pseudo-estado) para o outro. Sua representação é uma seta de ponta aberta , rotulada com o identificador da transição, descrito a seguir:

  <gatilho>[‘,’<gatilho >][‘[‘<guarda>’]’][‘/’<efeito>]* ,

  onde:

  <gatilho> é o evento que provoca a transição de estado;

<*guarda*> é a função *booleana* que estabelece uma condição para que a transição ocorra. Assim, quando o evento associado ao gatilho ocorre, a transição só acontecerá se a guarda resultar *True*, e <efeito> é uma atividade executada exatamente antes da conclusão da transição de estado.

Outros dois elementos sintáticos são utilizados para especificar o início e o final de uma evolução de estados:

- estado final: representa uma situação de uma evolução de estados, da qual não é possível sair. Um diagrama pode apresentar vários estados finais, assim como nenhum. É representado por um pequeno círculo cheio dentro de um círculo vazio;
- pseudo-estado inicial: corresponde a um ponto de partida de uma evolução de estados. Denota o início do processamento modelado e não é caracterizado como um estado. Um diagrama deve apresentar exatamente um pseudo-estado inicial, de onde as transições apenas saem. É representado por um pequeno círculo cheio.

A figura 2.6 ilustra um exemplo do diagrama de máquina de estados.

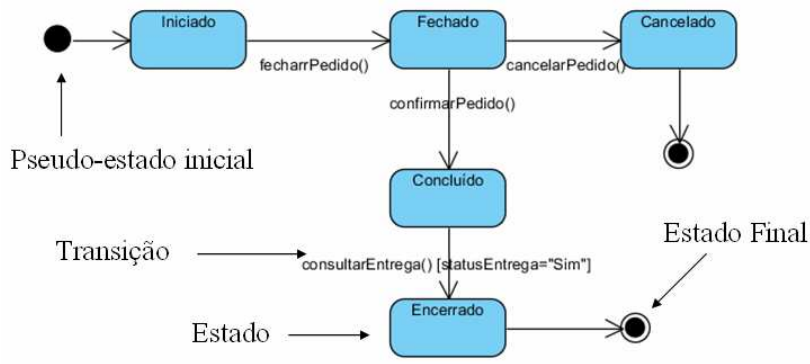


Figura 2.6: Diagrama de máquina de estados.

Conforme Pender (2004), existem ainda elementos utilizados para permitir uma série de opções para especificar situações complexas:

- pseudo-estado de bifurcação (*fork*): permite que uma única transição se divida em diversos caminhos percorridos concorrentemente.

- pseudo-estado de união (*join*): modela uma situação onde a máquina de estados espera que todas as transições de entrada cheguem, antes de disparar a transição de saída. Sincroniza fluxos concorrentes.
- pseudo-estado de junção (*junction*): corresponde ao desvio condicional estático, oferecendo um meio para simplificar condições de guarda compostas, combinando as partes semelhantes em um único segmento de transição.
- pseudo-estado de escolha (*choice*): corresponde ao desvio condicional dinâmico, oferecendo meios para isolar um comportamento do seu efeito. O comportamento forma o primeiro segmento da transição, antes do pseudo-estado de escolha. As alternativas, baseadas no resultado do comportamento, formam os segmentos de saída após e pseudo-estado de escolha.

No contexto da modelagem de software orientado a objetos, a principal finalidade do diagrama de máquina de estados é a descrição da existência de um objeto, instância de uma classe. Outros usos, porém, também são previstos como por exemplo a modelagem de algoritmo de método de classe.

No contexto da modelagem de software orientado a componentes, o diagrama de máquina de estados pode ser utilizado para modelar o comportamento dos componentes. É nesse sentido que esse trabalho o utiliza.

2.2.5 Diagrama de Implantação (*deployment diagram*)

> O diagrama de implantação, cuja denominação original é deployment diagram, é voltado a mostrar a organização do conjunto de elementos de um sistema computacional para que ocorra sua execução (SILVA, 2007, p.90).

É composto por um conjunto de nodos e associações entre eles. Os elementos sintáticos do diagrama de implantação são:

- nodo: nodos são unidades de processamentos conectados entre si para permitir a comunicação entre os artefatos que residem neles (PENDER, 2004). Em um diagrama, podem ser representados tantos os nodos como instâncias de nodos, que correspondem a uma ocorrência concreta de um nodo (SILVA,

2009). É representado, geralmente, por um retângulo tridimensional, mas podem ser utilizados ícones alternativos.

- Caminhos de comunicação: representam a conexão entre os nodos. As associações entre instâncias de nodos correspondem a ligações que estabelecem caminhos de comunicação entre esses elementos.
- artefato: "um artefato representa um elemento físico que corresponde a uma fração de informação de um sistema operacional, que pode ser disposto (*deployed*) em um nodo". (SILVA, 2007, p.222). Ocorrências de artefatos, representando instâncias de componentes, podem ser dispostas em instâncias de nodos para representar a composição de um sistema computacional. Sua notação é um retângulo com um ícone de arquivo no canto superior direito, mas pode-se usar simplesmente um retângulo com o estereótipo <<*artifact*>> ou algum outro ícone.

A figura 2.7 ilustra um exemplo de diagrama de implantação.

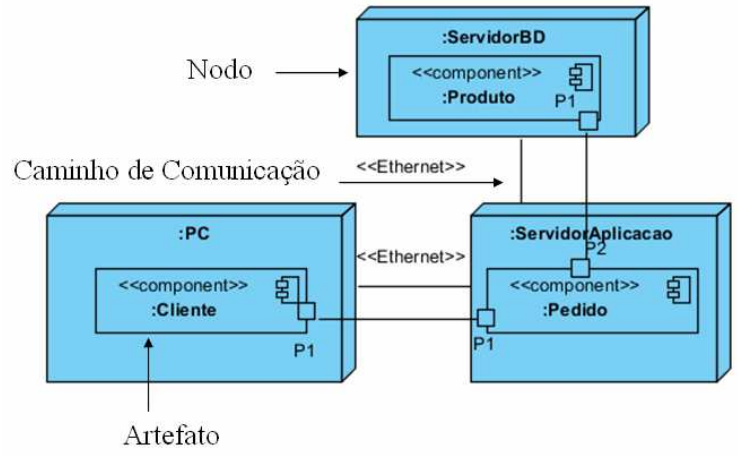


Figura 2.7: Diagrama de implantação.

Nesse trabalho o diagrama de implantação é utilizado para descrever a organização dos componentes interligados de um sistema orientado a componentes.

2.3 REDES DE PETRI

Redes de Petri foram criadas a partir da tese de doutorado de Carl Adam Petri, intitulada *Kommunication mit Automaten* (Comunicação com Autômatos), apresentada à Universidade de Bonn em 1962. Rede de Petri é uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática (MACIEL, LINS e CUNHA, 1996). Essa técnica possui a particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determiníticos. A aplicação das redes de Petri atingiu áreas como a modelagem de componentes de hardware e software, controle de processos, linguagens de programação, sistemas distribuídos e protocolos de comunicação.

Nessa seção são tratados os conceitos básicos relacionados às redes de Petri, assim como as técnicas de análises utilizadas para identificar propriedades na rede. Algumas dessas análises são consideradas nesse trabalho para a identificação de possíveis problemas comportamentais na interação de componentes.

2.3.1 Definição

Os elementos básicos que formam a estrutura topológica das redes de Petri são: lugares; transições e arcos.

Os lugares são usados para modelar os elementos passivos dos sistemas, isto é, correspondem às suas variáveis de estado, formando um conjunto P = {p1, p2, ..., pn}.

As transições são usadas para modelar os elementos ativos dos sistemas, ou seja, os eventos que levam o sistema de um estado a outro, formando um conjunto T = {t1, t2, ..., tm}.

Os arcos são usados para especificar como se dá a transformação de um estado em outro pela ocorrência das transições no sistema.

Graficamente, os elementos são representados da seguinte forma:

- Lugares : círculos ou elipses 
- Transições: barras
- Arcos: setas 

Matematicamente, conforme Cardoso e Valette (1997), uma rede de Petri é uma quádrupla: $R=<P,T,Pré,Post>$, onde:

- *P* é um conjunto finito de lugares de dimensão *n*;
- *T* é um conjunto finito de transições de dimensão *m*;
- *Pré* é a função de entrada (identifica todos os lugares de entrada de uma transição);
- *Post* é a função de saída (identifica todos os lugares de saída de uma transição).

Os lugares de uma rede de Petri podem ser marcados com fichas (*tokens*) e aos arcos pode ser associado um peso. *M(p)* é o número de fichas contidas no lugar *p*. A marcação *M* é a distribuição de fichas nos lugares de uma rede, sendo representada por um vetor coluna cuja dimensão é o número de lugares e elementos *M(p)*.

O estado de uma rede de Petri é definido por sua marcação atual *M*. Uma rede de Petri marcada *N* é uma dupla: $N = <R,M>$, onde:

- *R* é uma rede de Petri,
- *M* é uma marcação;

Uma transição está sensibilizada ou habilitada, isto é, pode ser disparada, quando o número de fichas em cada um dos lugares de entrada for maior ou igual ao peso do arco que liga este lugar à transição.

O disparo de uma transição t é uma operação que consiste em retirar fichas dos lugares de entrada e colocá-las nos lugares de saída, modificando a marcação da rede. Isso significa a mudança de estado do sistema representado, devido a ocorrência do evento associado a transição t.

O conjunto de marcações acessíveis de uma rede de Petri marcada é o conjunto das marcações que podem ser atingidas a partir da marcação inicial, através de uma sequência de disparos.

2.3.2 Classificação das redes de Petri

Uma das maneiras mais utilizadas pela literatura para classificar as redes de Petri é quanto ao tipo de marcação. Nesse caso pode-se agrupá-las em duas grandes classes: as ordinárias (de baixo nível) e não-ordinárias (de alto nível).

As redes ordinárias são aquelas cujos significados de suas marcas não são diferenciáveis, podendo ser do tipo inteiro e não negativo,

representando condições *booleanas*. As redes ordinárias se subdividem em:

- Rede Binária: é a rede mais elementar dentre todas. Essa rede só permite no máximo um *token* (ficha) em cada lugar, e todos os arcos possuem valor unitário.
- Rede *Place-Transition* (Lugar-Transição): é o tipo de rede que permite o acúmulo de fichas no mesmo lugar, assim como valores não unitários para os arcos.

As redes não-ordinárias são aquelas que possuem marcas de tipos particulares, que incorporam alguma semântica, viabilizando sua diferenciação. Esta semântica pode ir desde a atribuição de valores ou cores às marcas, até a adoção de noções de tipos de dados abstratos, conferindo-lhes um grande poder de expressão. No presente trabalho são usadas as redes de Petri ordinárias e binárias, conforme apresentado no capítuo 3.

2.3.3 Propriedades das redes de Petri

Segundo Cardoso e Valette (1997), as propriedades relativas à rede de Petri podem ser divididas em dois grupos:

1. Propriedades relativas às redes de Petri marcadas: suas definições implicam considerações sobre o conjunto de marcações acessíveis a partir da marcação inicial. O método de análise dessas propriedades é baseado no grafo de marcações e é apresentado na próxima seção.
2. Propriedades relativas às redes de Petri não marcadas (ou propriedades estruturais): independem de sua marcação inicial e permitem derivar métodos de cálculo, diretamente das definições, através da resolução de um sistema de equações lineares.

2.3.3.1 Propriedades relativas às redes marcadas

A seguir são apresentadas as propriedades relativas às redes marcadas.

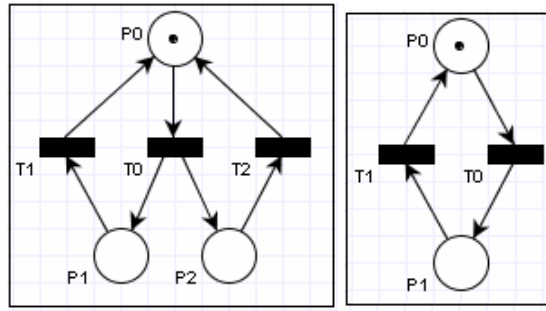
- Limitada (*bounded*): Uma rede é limitada quando não há surgimento indefinido de fichas em nenhum lugar da rede.

> "Este conceito corresponde ao fato de que um sistema físico é sempre limitado. Entretanto, uma rede de Petri não limitada pode ser utilizada quando se deseja avaliar o desempenho de um sistema independentemente de seus elementos de armazenamento intermediários" (CARDOSO e VALETTE, 1997, p.63).

A figura 2.8.a apresenta um exemplo de uma rede de Petri não limitada e a figura 2.8.b, uma limitada.

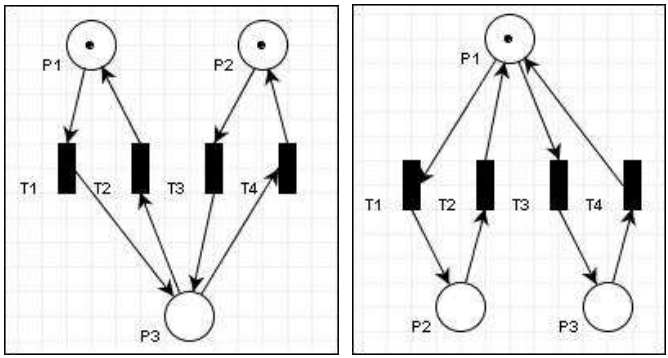
- Binária, salva ou segura (s*afe*): Uma rede é binária se todos os lugares contiverem no máximo uma única ficha. Lugares com apenas uma ficha representam condições *booleanas* (verdadeiro ou falso). "Se os lugares representam condições lógicas, a presença de mais de uma ficha num lugar significa que uma incoerência foi introduzida no modelo" (CARDOSO e VALETTE, 1997, p.63). Para alguns contextos, pode ser que lugares não representem condições deste tipo, permitindo, portanto, que eles tenham mais de uma ficha.

A figura 2.9.a apresenta um exemplo de uma rede de Petri não binária e a figura 2.9.b, uma rede de Petri binária.



a) b)

Figura 2.8: Rede de Petri: a) não limitada; b) limitada.

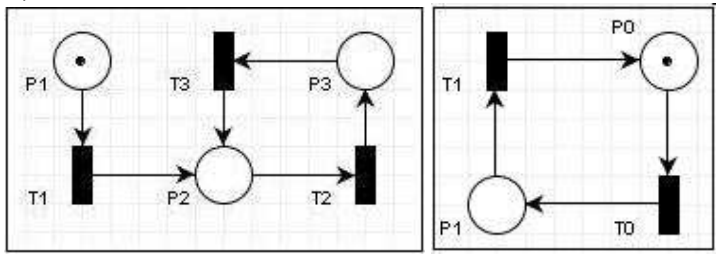


a) b)

Figura 2.9: Rede de Petri: a) não binária; b) binária.

- Reiniciável: Uma rede é reiniciável (ou própria) quando a partir de qualquer marcação, realizando-se uma sequência de disparos de transição, chega-se à marcação inicial.

Nos sistemas que possuem funcionamento repetitivo, as redes de Petri utilizadas para representá-los são reiniciáveis. A figura 2.10.a apresenta um exemplo de uma rede de Petri não reiniciável e a figura 2.10.b, uma rede de Petri reiniciável.



a) b)

Figura 2.10: Rede de Petri: a) não reiniciável; b) reiniciável.

- Viva: Uma rede é viva se todas as suas transições forem vivas. Uma transição em uma rede de Petri pode ser viva, quase viva ou morta.

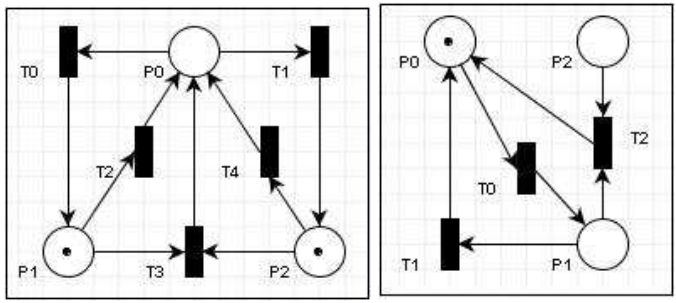
Uma transição é viva se ela pode ser habilitada a partir de qualquer marcação acessível através de uma sequência de disparos.

Uma transição é quase viva quando ao ser disparada, uma ou mais vezes, não é mais habilitada, impossibilitando assim seu disparo em um dado momento.

Uma transição é considerada morta quando ela nunca será habilitada, isto é, nunca será atingida.

"Uma rede viva garante que nenhum bloqueio pode ser provocado pela estrutura da rede, e garante também a ausência de partes mortas (nunca atingidas)" (CARDOSO e VALETTE, 1997, p.66).

A figura 2.11.a apresenta um exemplo de transição quase viva e a figura 2.11.b de uma transição morta. Na figura 2.11.a, a transição "T3" ao ser disparada jamais se torna habilitada novamente. Na figura 2.11.b, a transição "T2" nunca será disparada, já que não há como o lugar "P2" receber ficha.



a) b)

Figura 2.11: Transição: a) quase viva; b) transição morta.

- Bloqueio (*deadlock*): Uma rede está bloqueada quando em sua evolução não existe nenhuma transição habilitada, isto é, que possa ser disparada.

A figura 2.12 apresenta um exemplo de uma rede com bloqueio.

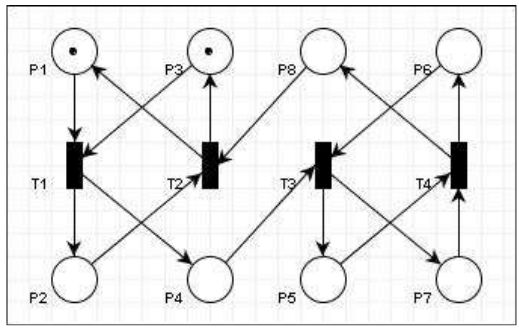


Figura 2.12: Rede de Petri bloqueada.

2.3.3.2 Propriedades relativas às redes não marcadas

As propriedades relativas às redes não marcadas, ou estruturais, são definidas através dos componentes conservativos de lugar e dos componentes repetitivos estacionários. A partir desses elementos estruturais, pode-se utilizar também a informação sobre a marcação, definindo-se assim, os invariantes de lugar e de transição, que permitem algumas informações adicionais sobre o comportamento dinâmico da rede de Petri.

- **Componentes conservativos, invariantes de lugar**: O invariante de lugar indica que para qualquer marcação acessível a partir da marcação inicial, a soma das fichas desses lugares será sempre constante.

- **Componentes repetitivos, invariantes de transição**: O invariante de transição corresponde a uma sequência cíclica de eventos que pode ser repetida indefinidamente. O disparo do conjunto de transições de um invariante de transição não modifica a marcação da rede. A sequência das transições "T0" e "T1" da figura 2.8 (rede de Petri limitada) é um invariante de transição. O conjunto das transições "T0"e "T1" do invariante forma um componente repetitivo estacionário da rede.

2.3.4 Análise das redes de Petri

Os métodos de análises foram desenvolvidos para verificar o comportamento de uma rede de Petri, identificando se ela possui ou não uma determinada propriedade. Cardoso e Valette (1997) classificam esses métodos em três grupos.

1. Análise por enumeração das marcações: esse método é válido somente para as redes marcadas. Ele simula a execução da rede de Petri através da árvore de cobertura. Parte-se da marcação inicial e cada transição sensibilizada por esta marcação dá origem a um ramo. As marcações obtidas através do disparo das transições são calculadas e o processo recomeça para cada marcação obtida. Para as redes de Petri limitadas, a árvore de cobertura é chamada de árvore de alcançabilidade, que contém todas as marcações possíveis. Somente nesse caso é possível encontrar todas as possibilidades.

"A vantagem deste método é que, se for possível construir o grafo, todas as propriedades podem ser encontradas". (CARDOSO e VALETTE, 1997, p.76).

2. Análise estrutural: essa análise é baseada na equação fundamental das redes de Petri e considera os componentes repetitivos e conservativos. A vantagem é que os resultados são válidos para qualquer marcação. "Os invariantes de lugar e transição, obtidos, respectivamente, a partir dos componentes conservativos e repetitivos estacionários, dependem da marcação inicial" (CARDOSO e VALETTE, 1997, p.80).
3. Análise por redução: esse método consiste na aplicação de regras de redução, de modo que a rede de Petri inicial e a rede reduzida sejam equivalentes do ponto de vista das propriedades. Após a redução, aplica-se um dos métodos descritos anteriormente. "A noção de equivalência não implica necessariamente que as duas redes possuam as mesmas sequências de disparo de transições ou os mesmos invariantes de lugar e de transição inicial". (CARDOSO e VALETTE, 1997, p.85). Por esse motivo, essa análise não será tratada neste trabalho, pois essas informações são tratadas no estudo do comportamento da interligação de componentes. Neste trabalho, somente as análises 1 e 2 são realizadas.

2.4 OCEAN/SEA

Nesta seção, é apresentado o histórico, desde sua concepção até a versão atual, do *framework* OCEAN e do ambiente SEA, no qual foram inseridas ferramentas para avaliação desse trabalho.

2.4.1 O que é OCEAN/SEA

OCEAN é o nome do *framework* concebido por SILVA (2000) em sua tese de doutorado intitulada "Suporte ao Desenvolvimento e Uso de Frameworks e Componentes" e apresentada à Universidade Federal do Rio Grande do Sul em 2000. O OCEAN possibilita, dentre outras coisas, a criação de ambientes de desenvolvimento de software. Este *framework* deu origem ao ambiente SEA, que provê suporte para o desenvolvimento de software orientado a objetos.

O *framework* OCEAN e, consequentemente, o ambiente SEA, foram implementados originalmente em SMALLTALK, sob o ambiente VisualWorks, reutilizando classes da biblioteca de classes deste ambiente.

SEA é uma extensão da estrutura de classes do *framework* OCEAN e foi criado para manipular diferentes estruturas de especificação e funcionalidades. Em sua forma original, o ambiente já fornecia suporte à especificação e ao desenvolvimento de *frameworks*, componentes e aplicações, com estruturas baseadas no paradigma de orientação a objetos, possibilitando a utilização integrada dessas abordagens. Para tanto, era necessária a construção das especificações de projeto destes artefatos. Essas especificações eram feitas basicamente em UML, com extensões propostas pelo autor para poder especificar as características de flexibilidade dos projetos que a UML, em sua versão anterior a 2.0, não conseguia suprir. Alguns exemplos de especificações que não existiam em UML e que foram usadas no SEA em sua versão original é a Interface de componentes (com explicitação de portos, até então chamados de canais). Outras especificações, não contempladas pela UML e tratados no ambiente SEA, são o modelo de rede de Petri e diagrama de corpo de método.

2.4.2 A reengenharia do OCEAN/SEA

Em meados de 2005, o *framework* OCEAN passou por uma reengenharia onde foi reestruturado e recodificado em linguagem Java[3], por meio do trabalho de Coelho (2007), que teve como foco principal a recodificação do núcleo do *framework*. Porém, o autor propôs melhorias na nova versão para facilitar a sua manutenção. Como o *framework* OCEAN não é um artefato executável, foi necessário usar uma aplicação para validar a reengenharia. Para isto, foi convertido, para a linguagem Java, também parte do ambiente SEA, assim como a estrutura e todos os elementos de uma especificação orientada a objetos. O ambiente SEA usa a totalidade dos recursos do *framework* OCEAN, fato que justificou a sua escolha. Já a especificação orientada a objetos foi escolhida por ser a especificação mais usual e a que possuía o maior número de modelos e conceitos das especificações previstas no ambiente SEA.

[3] A especificação da linguagem Java está disponível em: http://www.java.com/pt_BR/

Desta forma, a reengenharia do *framework* OCEAN foi validada através do uso de uma especificação OO no ambiente SEA. Para tanto, fez-se uso da interface gráfica projetada para o ambiente SEA, fruto do trabalho de Machado (2007) que realizou a reengenharia do suporte à construção de interfaces do *framework* OCEAN.

O *framework* OCEAN foi construído para atender à criação de ambientes de desenvolvimento de software. Portanto, foi necessário prover suporte à edição gráfica, um meio pelo qual pudessem ser construídos os diagramas. Para fornecer esta característica, utilizou-se originalmente o HotDraw, *framework* escrito em Smalltalk que auxilia no desenvolvimento de editores gráficos. Durante a reengenharia do *framework* OCEAN para a linguagem Java, igualmente havia demanda de suporte gráfico. Entretanto, para isto, utilizou-se a versão Java do HotDraw, o *framework* JHotDraw, embutido no OCEAN por Amorim (2006) em seu trabalho de conclusão de curso.

O JHotDraw é um *framework* que auxilia no desenho técnico e gráfico estruturado de aplicações em linguagem Java. É voltado para o desenvolvimento de editores gráficos bidimensionais com semântica associada aos elementos gráficos.

A existência do JHotDraw foi um dos fatores que motivou o uso da linguagem Java, pois era necessário existir um artefato na linguagem escolhida que possibilitasse a criação de editores gráficos, para que pudesse servir como artefato equivalente ao HotDraw implementado em SmallTalk.

2.4.3 A segunda versão em Java do ambiente SEA

O ambiente SEA, em sua primeira versão em Java, não utilizou todas as técnicas de modelagem da UML, pois o suporte a diagramas nessa versão foi uma atividade secundária, visto que o objetivo de se criarem diagramas era validar o *framework* a fim de que futuros pesquisadores pudessem usufruí-lo de maneira adequada. Assim, poucos diagramas foram suportados e alguns ficaram incompletos. Além disso, detalhes estéticos não foram observados.

Dentro os diagramas suportados nessa primeira versão em Java do ambiente SEA estão:

1. Diagrama de Casos de Uso: os relacionamentos de extensão e inclusão não foram implementados.
2. Diagrama de Classes: com as funcionalidades básicas supridas.

3. Diagrama de Sequência: com as funcionalidades básicas supridas.
4. Diagrama de Atividades: apenas com a estrutura base e o conceito de atividade criados.
5. Diagrama de Transição de Estados: apenas com a estrutura base e o conceito de estado criados.
6. Diagrama de Corpo de Método: concebido por SILVA (2000), oferece suporte à modelagem de algoritmo de métodos de classe. Não é prevista em UML.

Para suprir essa carência, Vargas (2008) deu suporte à edição dos diagramas da segunda versão de UML (UML2) no ambiente SEA, tornando o ambiente mais aplicável e adequado para utilização. A especificação UML já existente foi estendida para integrar os novos diagramas da UML2. Desta forma, o ambiente SEA ganhou sua segunda versão na linguagem Java.

Dos treze diagramas pertencentes à UML2 na época, a nova versão Java do ambiente SEA passou a dar suporte a onze diagramas. Dos diagramas que já existiam anteriormente, o único que não foi criado ou modificado significativamente foi o diagrama de classes, por já apresentar as características básicas. Assim, dois diagramas inéditos da UML2 não foram cobertos por Vargas (2008): o Diagrama de Temporização e o Diagrama de Visão Geral de Interação. Todos os outros diagramas foram modificados e/ou implementados, com exceção do diagrama de Perfil, não existente na época.

A figura 2.13 ilustra a tela principal do ambiente SEA, com um diagrama de componentes em edição.

Com o objetivo de suportar a produção de especificações orientadas a objetos, o ambiente SEA original também apresentava um conjunto de funcionalidades que podem estar incorporadas às ferramentas (COELHO, 2007). Essas funcionalidades são: avaliação de consistência de especificações OO; alteração de relações semânticas; reuso de conceitos por cópia e colagem; criação automática de métodos de acesso aos atributos; suporte à composição do diagrama de corpo de métodos; suporte para alteração de *frameworks*; suporte a padrões de projeto e suporte a engenharia reversa e geração de código. Apenas parte dessas funcionalidades está disponível na versão Java.

O *framework* OCEAN e a segunda versão do ambiente SEA em Java proporcionam a realização da avaliação da abordagem proposta neste trabalho, visto que já possui todos os diagramas da UML2, além da possibilidade do desenvolvimento e inclusão de novas ferramentas no ambiente.

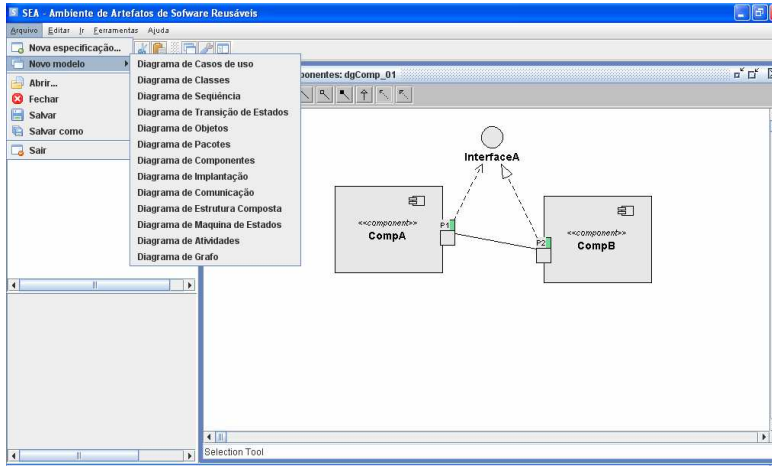


Figura 2.13: Tela principal de edição do ambiente SEA versão 2.0.

2.5 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Esse capítulo apresentou conceitos e técnicas necessários para o entendimento da abordagem proposta. Foram apresentados os conceitos relacionados ao desenvolvimento de software baseado em componentes. Nesse trabalho, componentes são especificados para gerar novos artefatos de software. A descrição dos componentes é feita utilizando a UML, por meio da proposta de Silva (2009), que é apresentada no capítulo 4. Os diagramas e seus elementos sintáticos, utilizados para a especificação do software baseado em componentes, foram descritos.

A proposta deste trabalho realiza a conversão do modelo UML para redes de Petri, que é um modelo formal. Isso possibilita análises automáticas sobre a especificação para identificar possíveis incompatibilidades entre os componentes interligados em uma aplicação. As redes de Petri, suas propriedades e as possíveis formas de análises foram mostradas nesse capítulo.

Por fim, foi dado um breve histórico do *framework* OCEAN e do ambiente SEA, utilizados para desenvolver a abordagem apresentada nesse trabalho.

3 TRABALHOS CORRELATOS

Várias abordagens têm sido sugeridas em pesquisas para a especificação da interface de componentes, na tentativa de proporcionar meios que permitam a verificação de compatibilidade entre componentes interligados.

Esse capítulo apresenta trabalhos que sugerem formas para a descrição da interface de componentes, assim como métodos aplicados para análise de compatibilidade. Essas propostas incluem um conjunto de combinações de técnicas utilizadas para esses propósitos.

Também é realizada uma análise de algumas pesquisas que utilizam linguagens formais para fornecer uma semântica formal para as notações UML, através da conversão de modelos UML para um modelo formal.

Ao final, os trabalhos correlatos são comparados com esta pesquisa.

3.1 APRESENTAÇÃO DOS TRABALHOS CORRELATOS

Nessa seção, são brevemente apresentados os trabalhos correlacionados a esta dissertação.

Os trabalhos selecionados se restringem à língua inglesa e portuguesa. A primeira por abranger a maioria das publicações e a segunda por muitas vezes apresentar a realidade brasileira. As fontes de busca incluíram *journals* e *proceedings* do IEEE, ACM, *GoogleScholar*, periódicos presentes no portal CAPES, congressos relacionados à área de pesquisa, bem como outros *journals* em outras áreas.

A revisão bibliográfica aplicada nesse trabalho se deu por meio da identificação, coleta e análise de fontes literárias relacionadas aos seguintes temas:

1. Desenvolvimento de software orientado a componentes.
2. Compatibilidade de componentes.
3. Compatibilidade estrutural de componentes.
4. Compatibilidade comportamental de componentes.
5. Problemas relacionados à compatibilidade de componentes.
6. Redes de Petri e
7. UML.

As palavras chaves utilizadas na pesquisa foram: "*component-based software development*", "*component-based software engineering*", "*component-oriented software development*", "*compatibility of components*", "*structural compatibility of components*", "*compatibility behavioral components*", "*problems related to compatibility of components*", "*Petri net and component of software*", "*UML and component of software*", "*UML and formal models*", "*using the UML in the development of component-based software*", suas respectivas traduções para o português e palavras correlatas.

A seleção dos trabalhos foi baseada nos seguintes critérios:

1. O trabalho aborda o desenvolvimento de software orientado a componentes em nível de especificação.
2. O trabalho trata a análise de compatibilidade estrutural, comportamental ou ambos.
3. O trabalho aborda redes de Petri no contexto dos componentes.
4. O trabalho aborda a UML no contexto dos componentes.
5. O trabalho aborda a conversão da UML para um modelo formal.

Os trabalhos apresentados da seção 3.1.1 à seção 3.1.6 tratam da avaliação de compatibilidade de componentes em nível de especificação. Porém, cada um deles adota diferentes técnicas tanto para a especificação dos componentes como para a avaliação de compatibilidade entre eles. A seção 3.1.7 discute a proposta para especificação de componentes utilizando UML, adotada neste trabalho e apresentada em detalhes no Capítulo 4. A seção 3.2.1 referencia trabalhos que tratam da conversão da UML para algum modelo formal e a seção 3.2.2 discute trabalhos que utilizam redes de Petri para obeter esse formalismo.

Todos esses trabalhos referenciados contribuíram para essa dissertação.

3.1.1 *Software Architecture Analysis based on Statechart Semantics* (DIAS e VIEIRA, 2000)

A proposta de Dias e Vieira (2000) utiliza ADL (*Architecture Description Language*) para descrever o aspecto estrutural da interface

de componente e o diagrama de estados da UML para representar o aspecto comportamental.

A abordagem é suportada por uma ferramenta chamada Argus-I, que fornece suporte para as análises que são realizadas e que, segundo os autores, abrangem tanto o nível individual dos componentes como arquitetural.

No nível individual, é realizada a análise estrutural de um componente, em que uma coleção de regras (*type checking*) é aplicada sobre a especificação do componente para determinar se ele está bem especificado, isto é, se atende às regras, ou não. Para a análise comportamental do componente, é feita a conversão do diagrama de estados para a linguagem PROMELA (PROMELA, 2011), e então a ferramenta SPIN[4], integrada a ferramenta Argus-I, é usada para verificar a consistência do modelo.

No nível arquitetural são realizadas análises de dependência e simulação. Na análise de dependência são identificadas interfaces incompatíveis. Para a simulação, é gerado um "super modelo de estado", integrando todos os diagramas de máquina de estados dos componentes. Ele também é convertido para a linguagem PROMELA (PROMELA, 2011) e analisado pela ferramenta SPIN. Um erro é detectado quando um componente recebe um evento não esperado.

A idéia de representar o comportamento da aplicação por meio da união de todos os componentes envolvidos já havia sido proposta por Silva e Price (1999) em uma publicação anterior. Porém, a representação do comportamento nessa abordagem era feita em redes de Petri como é visto na seção 3.1.4.

Dias e Vieira (2000) não definem como o super modelo de estados - que representa o modelo comportamental dos componentes interligados - é gerado e como as análises são realizadas sobre ele. As regras utilizadas para a análise estrutural de cada componente individual também não são esclarecidas, assim como as verificações feitas pela ferramenta SPIN. De um modo geral, o artigo não apresenta evidências de que a abordagem proposta tenha uma solução bem definida. Os problemas detectados nas análises são mais restritos se comparados com o conjunto de problemas identificados na proposta desta dissertação.

A proposta de utilizar ADL e diagrama de estados da UML para descrever a interface de componentes foi uma tentativa de suprir as

[4] SPIN é uma ferramenta de análise de modelos descritos em PROMELA que se destina a verificação formal de sistemas concorrentes. Disponível em http://spinroot.com/spin/whatispin.html.

lacunas existentes nas duas técnicas. Segundo os autores, as ADLs possuem limitações na definição semântica, pois não conseguem representar de forma completa e precisa aspectos comportamentais, e a UML não representa adequadamente conceitos arquiteturais.

De fato, a UML até a versão 1.5 (OMG, 2003) possuía limitações em representar conceitos de componentes e sistemas baseados em componentes. A partir da versão 2.0 (OMG, 2005), publicada em 2005, essas limitações foram supridas com a incorporação de diagramas e elementos que representam de forma precisa tais conceitos, como extensões ao diagrama de componentes e a inclusão do diagrama de estrutura composta. (MOUAKHER, LANOIX e SOUQUIÈRES, 2006) e (SILVA, 2009).

Abordagens que utilizam mais de uma técnica para especificar sistemas baseados em componentes exigem que os projetistas conheçam ambas, o que pode demandar maior esforço.

3.1.2 *Behavior Protocols for Software Components* (PLASIL e VISNOVSKY, 2002)

Outra forma possível de descrever o comportamento de componentes é através de protocolos de comportamento (*Behavior Protocol*), baseados em expressão regular e propostos por Plasil e Visnovsky (2002). Um protocolo de comportamento é sintaticamente composto por eventos e operadores. As análises, considerando essa abordagem, verificam duas situações:

- quando um componente (de acordo com seu protocolo de comportamento) pode emitir um evento, mas não há outro componente capaz de aceitar tal evento, e;
- *deadlock*, ou seja, o estado onde nenhum componente é capaz de executar nenhuma ação (nem emitir nem receber um evento) e o componente não está no seu estado final.

A abordagem não trata como os componentes são conectados para formar uma aplicação. Portanto, problemas relacionados com o conjunto de componentes conectados não são identificados. As análises são realizadas a partir da especificação comportamental de cada componente individualmente. Dessa forma, a gama de verificações realizadas nessa abordagem é mais limitada se comparada com as verificações realizadas no presente trabalho. A análise estrutural não fez parte do escopo da proposta, que teve como foco, o aspecto comportamental.

A verificação de compatibilidade entre componentes através de especificações utilizando protocolos de comportamento mostrou-se problemática. Kofron (2007) identificou problemas relacionados a explosão de estados na ferramenta de verificação que utiliza protocolos de comportamento (KROFON, MACH e PLASIL, 2005). Para solucionar esse problema, Kofron (2007) propôs um método capaz de traduzir especificações feitas em protocolos de comportamento para a linguagem PROMELA (PROMELA, 2011). Assim, as verificações de compatibilidade entre componentes passam a ser realizadas utilizando a ferramenta SPIN. A motivação para o uso da ferramenta SPIN foi a sua capacidade de tratar tamanhos de *state space* numa ordem de grandeza maior do que aqueles tratados pela ferramenta que utiliza protocolos de comportamento.

3.1.3 *Component Adaptation: Specification and Verification* (MOUAKHER, LANOIX e SOUQUIÈRES, 2006).

O objetivo principal desse trabalho é propor adaptadores quando identificadas interfaces não compatíveis entre componentes. Para tanto, os autores utilizam uma abordagem baseada na proposta de Chouali e Souquières (2005).

Nessa abordagem a especificação da interface de componente consiste de um modelo de dados, descrito pelo diagrama de classe, e pelo protocolo da interface, modelado por meio do diagrama de estados, chamado de PSM (*Protocol State Machine*). Para cada operação são especificadas pré e pós-condições.

Essa especificação é então convertida para o método formal B (B Method, 2011), chamada, então, de máquina B do componente. O refinamento em B é usado para provar que duas interfaces são compatíveis por meio da ferramenta *Atelier B* (Atelier-B, 2011).

Uma das limitações desse trabalho é que as verificações realizadas não consideram o aspecto comportamental e os efeitos do conjunto das interligações de todos os componentes envolvidos na aplicação.

Além disso, a abordagem não trata o conceito "porto" de componente. Nesse caso, entende-se que o componente terá apenas um único ponto de conexão, o que restringe sua aplicabilidade. Assim, a representação da conexão entre componentes não se dá por meio de seus portos, mas por meio de uma interface requerida de um componente e uma interface fornecida do outro componente a ele conectado. Portanto,

as análises realizadas nesse trabalho são limitadas a duas interfaces: uma requerida e outra fornecida.

As limitações desse trabalho foram todas supridas pela proposta apresentada nesta dissertação.

A principal contribuição do trabalho sob análise para a abordagem sugerida nesta dissertação foi ratificar a deficiência da UML para suportar verificações formais e de interoperabilidade, mas por outro lado, evidenciar que a UML possui notações gráficas expressivas para especificar componentes e suas interfaces.

Adaptadores já haviam sido propostos em uma pesquisa publicada anteriormente (SARTORI, 2005), porém utilizando outra abordagem. O objetivo dos adaptadores é realizar a união entre dois ou mais componentes incompatíveis. Adaptadores estão fora do escopo desta dissertação.

3.1.4 Suporte ao Desenvolvimento e Uso de Frameworks e Componentes (SILVA, 2000)

Em sua tese de doutorado, Silva (2000) concebeu o framework OCEAN e o ambiente SEA, descritos no capítulo 2, na seção 2.4 OCEAN/SEA. Uma das contribuições desse trabalho foi uma nova forma de especificar componentes.

Nessa abordagem, os componentes são especificados como elementos de um diagrama de classes (que usam classes como fachadas dos componentes) associados a uma única IC, definida e selecionada em uma biblioteca de IC do ambiente. Um artefato baseado em componentes é produzido a partir da conexão de um conjunto de componentes, por meio de seus canais de conexão (como eram tratados os portos). A descrição estrutural da IC é feita definindo o relacionamento entre métodos (fornecidos e requeridos) e canais; a especificação comportamental, por meio do diagrama de redes de Petri. A especificação da interface de cada componente possui uma rede de Petri associada para representar o comportamento individual, isto é, as restrições de ordem de chamadas de serviços. Os lugares da rede representam pré ou pós-condições e as transições representam o par (porto/ método) do componente. Caso um método não esteja habilitado, isto é, as pré-condições não sejam satisfeitas, o método não poderá ser invocado. A proposta fez parte da primeira versão do ambiente SEA.

Ainda nessa proposta, foi concebida a idéia de representar o comportamento da arquitetura de componentes através da união das redes de Petri dos componentes individuais.

No modelo de especificação de arquitetura de componentes existe a conexão de diversos componentes para formar um sistema; então, se o comportamento de todos os componentes for unido em uma rede de Petri, respeitando-se as regras de conexões da especificação da arquitetura, o comportamento da aplicação estará representado (SILVA, 2000). Essa rede de Petri é chamada de rede resultante. Com essa rede montada expressando o comportamento da aplicação, análises sobre suas propriedades podem ser realizadas através de ferramentas. Dessa forma, é possível detectar e informar possíveis problemas decorrentes de um mau comportamento proveniente de alguma conexão como, por exemplo, o bloqueio do sistema quando um componente fica aguardando a liberação do recurso de outro componente. A maneira de obter a rede de Petri resultante consiste em fundir transições de porto/método correspondentes, isto é, portos conectados de diferentes componentes, onde um dos dois invoca o método e o outro, o implementa.

A maior contribuição da proposta de Silva (2000), além de inovação de como especificar a IC, é a possibilidade de se obter o comportamento da aplicação orientada a componentes através da junção das redes de Petri que representam o comportamento individual de cada componente conectado.

Porém, a especificação da IC requer mais de uma técnica de modelagem, já que é definida através do diagrama de interface de componente (baseado em UML) e pelo diagrama de rede de Petri. Essa especificação, formada por esses dois diagramas, é chamada de documento de especificação de interface (*Component Interface*). Isso exige que o especificador de software tenha conhecimento de ambas as técnicas. Além disso, modelos formais, apesar das vantagens do formalismo algébrico, não são comumente utilizados para especificar sistemas de software, devido ao seu baixo grau de abstração.

O fato dos componentes serem representados de forma semelhante a classes para UML causa uma certa confusão de conceitos, pois se trata de componentes caixas pretas, com canais de conexão e comportamento, e não simplesmente uma classe com métodos e atributos. A abordagem, apesar de possibilitar a análise de compatibilidade entre componentes, não avaliou quais os possíveis problemas que poderiam ser encontrados em uma aplicação baseada em componentes, além de *deadlock,*e não tratou formas de fazê-lo.

3.1.5 Suporte à Análise de Compatibilidade Comportamental e Estrutural entre Componentes no Ambiente SEA (CUNHA, 2005)

Em sua dissertação, Cunha (2005) propôs melhorias no suporte ao desenvolvimento e uso de componentes no ambiente SEA, tendo como foco principal a análise de compatibilidade de componentes, com ênfase na análise estrutural.

No ambiente SEA existiam algumas deficiências em relação à especificação de componentes, tais como: componentes representados como estruturas de classes, ausência da especificação para a arquitetura dos componentes e da representação visual dos mesmos. Além disso, faltava um mecanismo para unificação das redes de Petri e ferramentas de análises de compatibilidade de componentes.

Essas deficiências foram supridas com as propostas de Cunha (2005) e Sartori (2005). Juntos, os autores acrescentaram ao ambiente um editor gráfico para a especificação dos componentes e sua arquitetura, possibilitando a manipulação dos mesmos de forma visual. Além disso, ferramentas para análise de compatibilidade estrutural e comportamental em uma aplicação baseada em componentes foram propostas e desenvolvidas por Cunha (2005), enquanto Sartori (2005) incluiu a ferramenta de adaptação de componentes.

O editor para a representação da arquitetura de componentes, desenvolvido pelos autores no ambiente SEA, foi chamado de "Diagrama de Componente", diferente do diagrama com mesmo nome da versão então disponível na UML para representar componentes. No diagrama de componentes do ambiente SEA os componentes são representados pela sua interface, que consiste em uma agregação dos portos de comunicação, chamados canais (SILVA, 2000). A conexão entre os componentes se dá através da ligação entre os portos de dois deles. Apesar desse diagrama ter inserido o conceito de porto na representação do componente, ele tem uma representação gráfica diferente do diagrama de componentes da UML 2. Porém, a proposta de edição da arquitetura de componentes no ambiente SEA é semanticamente semelhante à forma de representação de componentes adotada na UML 2.

Um outro modelo, o de rede de Petri resultante, foi criado no ambiente SEA por Cunha (2005). O autor refinou e avaliou a idéia inicial de Silva (2000) para representar o comportamento de aplicações baseadas em componentes através da rede de Petri resultante (a partir da união das redes de Petri dos componentes conectados). Em sua

dissertação, Cunha (2005) construiu o mecanismo para a geração da rede de Petri resultante, possibilitando, assim, a realização de análises comportamentais sobre a aplicação de componentes.

Cunha (2005) também construiu ferramentas de análise estrutural e comportamental de componentes, com propostas de soluções que eram apresentadas ao usuário, permitindo-lhe escolher opções de resolução para compatibilidade de componentes. Os resultados dessas análises fornecem subsídios para a ferramenta de adaptação desenvolvida por Sartori (2005), que gera a especificação do componente de adaptação caso haja incompatibilidade entre os componentes interligados.

Toda a implementação das abordagens propostas por Silva (2000), Cunha (2005) e Sartori (2005) para o desenvolvimento de sistemas baseado em componentes no ambiente SEA em sua primeira versão, foram feitas na linguagem de programação SMALLTALK.

O trabalho de Cunha (2005) acrescentou ao ambiente SEA uma capacidade adicional, a especificação de sistemas baseados em componentes, já que possibilitou a geração da rede de Petri resultante para representar o comportamento da aplicação e a possibilidade de análises relacionadas à compatibilidade de componentes. A maior contribuição do trabalho de Cunha (2005) para essa dissertação foi a ratificação de que é possível a geração da especificação comportamental da aplicação baseada em componentes de forma automatizada para a realização de análises de compatibilidade.

Porém, essa abordagem ainda utiliza uma mistura de modelos para a especificação de software baseado em componentes. Para a representação da interface dos componentes e sua arquitetura utiliza um modelo baseado na UML. E para a representação do comportamento dos componentes e da aplicação utiliza o modelo formal rede de Petri. Isso exige que o desenvolvedor de software domine tais modelos.

Além disso, o trabalho teve como foco a análise de compatibilidade estrutural. A análise de compatibilidade comportamental é limitada a detectar apenas *deadlock* e não considera as outras propriedades das redes de Petri, descritas no capítulo 2, na seção 2.3.

3.1.6 Outras Abordagens utilizando redes de Petri.

Além dos trabalhos já descritos (SILVA, 2000) (CUNHA, 2005), outras abordagens utilizam redes de Petri para representar o

comportamento de componentes (CRAIG & ZUBEREK, 2007) (WEI e TONG, 2008).

Porém, nenhum desses trabalhos trouxe um avanço significativo se comparado com trabalhos publicados anteriormente.

A abordagem utilizada por Craig e Zuberek (2007) é muito semelhante à abordagem utilizada por Cunha (2005). A diferença é que os autores propuseram uma técnica mais eficiente para detectar *deadlock*, o único problema comportamental identificado na análise.

No trabalho de Wei e Tong (2008), uma extensão das Redes de Petri, chamada de *component network* (Rede de Componentes), é utilizada para representar o comportamento relativo a integração de componentes em uma aplicação. Porém, sem nenhuma inovação relevante.

Esses trabalhos foram importantes para essa pesquisa apenas para ratificar o uso das redes de Petri no contexto dos componentes. Porém, ficaram ausentes do quadro comparativo que será apresentado na seção 3.2.3 pela semelhança com trabalhos já presentes no quadro.

3.1.7 Como Modelar com UML 2 (SILVA, 2009)

Algumas mudanças importantes podem ser observadas na nova versão de UML, a partir da versão 2.0. Entre as alterações mais significativas, observa-se que o diagrama de componentes, através de recursos sintáticos introduzidos com o diagrama de estrutura composta, passa a poder descrever a estrutura interna de um componente. A impossibilidade de fazê-lo na primeira versão de UML tornava esse diagrama semanticamente pobre e com uma fraca ligação com os outros elementos de uma especificação orientada a objetos (SILVA, 2007).

Silva (2009) propõe convenções para a descrição de componentes e de software baseado em componentes, com os recursos sintáticos da UML 2. A proposta do autor utiliza exclusivamente diagramas da UML 2 para descrever os aspectos estrutural, comportamental e funcional.

Em resumo, nessa proposta a especificação estrutural utiliza o diagrama de componentes, de classes e de implantação. A especificação comportamental é tratada pelo diagrama de transição de estados e a funcional, pelo diagrama de caso de uso e o de atividades. O presente trabalho adota essa abordagem, tratada em detalhes no capítulo 4. A especificação funcional está fora do escopo desta dissertação.

Segundo o autor, a avaliação de compatibilidade comportamental envolve todo o conjunto de componentes interligados – diferente da

avaliação de compatibilidade estrutural, que trata um par de portos de cada vez. Por esse motivo, o autor propôs uma solução para gerar a especificação comportamental da aplicação.

Cada componente da aplicação tem sua própria máquina de estados e, uma vez conectados, passam a compor uma máquina de estados única, que é a máquina de estados da aplicação. A questão é se essa máquina de estados resultante da conexão das máquinas de estados dos componentes individuais consegue ter a evolução de estados esperada ou em algum momento fica bloqueada, sem possibilidade de proceder a novas transições. A máquina de estados da aplicação é produzida pela interligação das máquinas de estados individuais de cada componente envolvido. Produzida a máquina de estados da conexão de componentes, a próxima questão é como avaliar a compatibilidade comportamental, o que é feito a partir da avaliação da evolução de estados da máquina de estados da aplicação. Caso a evolução de estados seja possível, conclui-se que existe compatibilidade comportamental entre os componentes. Caso não seja possível a evolução de estados, a máquina fica bloqueada, o que caracteriza incompatibilidade comportamental (SILVA, 2009).

Diferente da abordagem de Dias e Vieira (2000), que não adota convenções e não deixa claro como o super modelo de estado é gerado, a proposta de Silva (2009) define quais os passos para compor a máquina de estados da aplicação, gerada pela interligação das máquinas de estados individuais de cada componente envolvido.

Além disso, a proposta de Silva (2009) tem bem definida a possibilidade de avaliação da compatibilidade estrutural e comportamental. Porém, ainda existem na proposta lacunas relacionadas às análises de compatibilidade.

Em se tratando da análise comportamental, o único problema mencionado na proposta foi o de bloqueio no sistema (*deadlock*), identificado a partir da evolução de estados da máquina de estados da aplicação. A proposta não define nem estabelece meios ou técnicas de como realizar essa análise.

Não existem ferramentas automatizadas para a especificação de componentes utilizando essa abordagem, o que dificulta sua avaliação e comprovação.

3.2 ABORDAGEM PROPOSTA PARA AVALIAÇÃO DA COMPATIBILIDADE DE COMPONENTES

O esforço de pesquisa aqui descrito realizou um estudo sobre as técnicas utilizadas para realizar a análise de compatibilidade de componentes. A partir dessa pesquisa, buscou-se definir uma estratégia para ser aplicada na realização das análises, tendo como foco a análise de compatibilidade comportamental, considerando a abordagem proposta, que utiliza exclusivamente diagramas da UML na especificação do software orientado a componentes proposta por Silva(2009). A estratégia definida deveria permitir a identificação de possíveis problemas comportamentais, além de bloqueio no sistema, e a automatização das análises de compatibilidade de componentes dentro do ambiente SEA.

3.2.1 Porque adotar uma abordagem utilizando UML

A UML é uma linguagem para especificar, visualizar e construir artefatos de sistemas de software. Ela é amplamente aceita pela comunidade de projetistas de software como uma notação padrão para a análise e projeto de sistemas de software orientado a objetos de fato. UML permite modelar diversos aspectos de sistemas complexos. Ela fornece os diagramas para a descrição dos aspectos estáticos, dinâmicos e arquitetônicos de sistemas com diferentes níveis de detalhe. Por esses motivos, avaliou-se ser vantajoso utilizar uma abordagem que utilize somente diagramas da UML para toda a especificação de sistemas baseado em componentes, assim como a proposta de Silva (2009).

Apesar da UML ser rica e expressiva, permitindo a visualização gráfica de modelos que facilitam a comunicação de ideias, ela foi desenvolvida sem considerar regras estritas para forçar o projetista a preparar uma especificação completa e consistente (OMG, 2011). Sua flexibilidade e extensibilidade são, na maioria das vezes, um grande atrativo. Mas também podem se tornar uma dificuldade quando há necessidade de realizar análises na especificação do sistema. Essa dificuldade advém da sua falta de formalismo, apesar da utilização de OCL (*Object Constraint Language*) (OMG, 2012). Mas é desejo da comunidade UML deixar certos aspectos semânticos em aberto, para apoiar uma maior utilização da notação em diferentes domínios. Ao mesmo tempo, uma semântica objetiva é essencial para apoiar verificações nos modelos.

Nesse caso, é possível atribuir uma semântica formal ao modelo UML para permitir validações automáticas (KING e POOLEY, 1999), (LILIUS e PALTOR, 1999), (MEYER e SOUQUIÈRES, 1999), (SALDHANA e SHATZ, 2000), (BARESI e PEZZÈ, 2001), (SCHÄFER, KNAPP e MERZ, 2001), (MOKHATI, GAGNON e BADRI, 2007), (THIERRY-MIEG e HILLAH, 2008) e (KONG et al, 2009).

3.2.2 Solução para a análise de compatibilidade de componentes

Considerando a abordagem de Silva (2009), a análise estrutural de componentes pode ser realizada utilizando os diagramas UML envolvidos na especificação da interface de componentes – diagrama de componentes, classes e implantação. A análise estrutural considera cada par de portos conectados dos diferentes componentes. No capítulo 4 essa análise é explicada em detalhes.

Já a análise comportamental é um pouco mais complexa, pois considera todo o conjunto de componentes interligados. A abordagem de Silva (2009) não estabelece como a máquina de estados da aplicação, descrita por meio do diagrama de estados da UML, deve ser analisada para a verificação da compatibilidade comportamental de componentes da aplicação.

O diagrama de máquina de estados é um modelo formal. Portanto, permite a verificação de propriedades do sistema. Porém, verificou-se nas pesquisas realizadas que outros modelos formais são mais comumente utilizados para essa finalidade, com técnicas de análise mais amadurecidas.

Para métodos formais, existe uma técnica genericamente chamada de verificação de modelos (*Model Checking*), que valida propriedades automaticamente em uma especificação associada ao problema. Esta técnica permite garantir que a especificação apresente as propriedades desejadas, como, por exemplo, a ausência de bloqueios no sistema (*deadlocks*). Verificação de modelos é feita através da enumeração exaustiva de todos os estados alcançáveis. Desta forma, pode-se assegurar que o software é livre de erro de projeto (para as propriedades verificadas) antes de implementá-lo. Essas verificações minimizam os riscos do projeto.

A utilização de ferramentas automatizadas que permitam identificar inconsistências e insuficiências nas fases iniciais do projeto (especificação é uma delas) é crucial para minimizar erros. Esses erros

podem causar prejuízos, sejam de tempo ou financeiro, nos projetos de software, uma vez que a experiência mostra que um percentual considerável dos acidentes relacionados à computação tem origem em erros de especificação (LEVESON, 1995). Especificações de sistemas devem ser verificadas já que podem conter comportamento não esperado pelo projetista.

A tabela 3.1 exibe, resumidamente, algumas propostas que convertem especificações UML em uma linguagem formal para realização de verificações automatizadas.

Tabela 3.1: Propostas de conversão da UML para modelo formal.

Ferramenta	Descrição	Referência
vUML	Converte o modelo UML para a linguagem PROMELA e invoca a ferramenta SPIN para a realização das análises.	(LILIUS e PALTOR, 1999) (SPIN, 2011) (PROMELA, 2011)
Argo UML	Ambiente com ferramentas para análise e projeto de sistemas orientado a objetos utilizando UML. Possui recursos cognitivos que fornecem aos projetistas de sistemas apoio à tomada de decisões. Esses recursos são inspirados em teorias da cognição humana.	(ROBBINS e REDMILES, 2000) (Argo UML, 2011)
Argus-I	Ferramenta implementada utilizando o *framework* desenvolvido por Argo UML. Converte o diagrama de transição de estados UML em PROMELA e utiliza a ferramenta SPIN para a realização das análises.	(DIAS e VIEIRA, 2000) (SPIN, 2011) (PROMELA, 2011)
HUGO	Projetado para verificar automaticamente se as interações expressas através de uma colaboração podem efetivamente ser realizadas por um conjunto de máquinas de estado. O modelo de estado é traduzido para PROMELA e o de colaborações para um conjunto de autômatos BÜCHI. A ferramenta SPIN é invocada para a verificação do modelo.	(SCHAFER, KNAPP e MERZ, 2001) (SPIN, 2011)
Método B	Método para transformação sistemática de especificações semi-formais expressas com notações OMT (*Object Modeling Technique)* em especificações formais em método B. A ferramenta Atelier-B[5] é utilizada para realizar a prova em B.	(MEYER e SOUQUIÈRES, 1999) (MOUAKHER, LANOIX e SOUQUIÈRES, 2006) (Atelier-B, 2011)
UML -	Framework para suporte à verificação formal de	(MOKHATI,

[5] Atelier B é uma ferramenta industrial que permite o uso operacional do método B para o desenvolvimento de software.

MAUDE	diagramas UML (classe, estado e comunicação) usando a linguagem Maude[6]. Realiza a tradução automática de diagramas de UML em uma especificação formal baseada em Maude e verifica algumas propriedades LTL usando o verificador de modelo integrado a Maude.	GAGNON e BADRI, 2007) MAUDE (2010)
UML - Rede de Petri	Abordagem para geração de modelos formais de rede de Petri a partir de modelos em UML.	(KING e POOLEY, 1999) (SALDHANA e SHATZ, 2000) (BARESI e PEZZE, 2001) (THIERRY-MIEG e HILLAH, 2008)

Um modelo formal bastante utilizado são as redes de Petri: uma linguagem formal gráfica capaz de modelar sistemas dinâmicos, com comportamento concorrente. Detalhes sobre as redes de Petri foram explicitados no capítulo 2. Algoritmos de análise podem ser aplicados sobre a rede, com o intuito de verificar as propriedades do sistema. É possível encontrar ferramentas prontas no mercado com essa finalidade (PIPE, 2011).

A estratégia proposta nesse trabalho para realizar a análise de compatibilidade comportamental entre componentes consiste em transformar a especificação comportamental de componentes (nessa abordagem tratada por meio do diagrama de máquina de estados da UML, este um modelo formal) em outro modelo formal mais facilmente analisável: redes de Petri.

A existência de ferramentas já consolidadas, inclusive com código aberto que permite integração com outras ferramentas, motivou a adoção das redes de Petri neste trabalho. Além disso, pesquisas anteriores do mesmo laboratório de pesquisas onde este trabalho foi realizado já haviam sugerido redes de Petri para a especificação comportamental de componentes (SILVA, 2000) e (CUNHA, 2005).

A característica da abordagem de utilizar exclusivamente UML não é afetada pela transformação do modelo UML em redes de Petri, visto que pode ser realizado de forma totalmente transparente para o projetista do sistema, através de uma ferramenta automatizada. O usuário da ferramenta pode não ter qualquer conhecimento sobre redes de Petri.

[6] A linguagem e sistema Maude é baseada na lógica de reescrita e permite a descrição formal de sistemas concorrentes.

Algumas pesquisas, como veremos a seguir, já abordam a conversão de modelos UML para redes de Petri.

A proposta de Saldhana e Shatz (2000) descreve uma metodologia para derivar o modelo de rede de Petri Objeto (*object Petri net*) do sistema a partir de diagramas de estados da UML. Nessa proposta, basicamente estados são mapeados em lugares e transições são mapeadas em transições. Os autores definem regras para tratar conceitos existentes no diagrama de máquina de estado.

Baresi e Pezzè (2001) indicam viabilidade de atribuir semântica formal à UML definindo regras que mapeiam automaticamente especificações UML para redes de Petri de alto nível. Os autores discutem as propriedades de especificações UML que podem ser traduzidas em modelos de rede de Petri de alto nível assim como as análises que podem ser realizadas.

Já na proposta de King e Pooley (1999) as traduções de especificações UML são feitas para redes de Petri estocásticas a fim de fornecer estimativas de desempenho.

O que essas propostas têm em comum é que, normalmente, se tratam de conversões tradicionais, onde, para o diagrama de máquina de estados, um estado é convertido em um lugar e uma transição (UML) corresponde e uma transição em rede de Petri.

Na abordagem desse trabalho, a especificação do comportamento da aplicação é feita através da união das máquinas de estados dos componentes envolvidos. Esse método foi proposto por Silva (2009) e é tratado em detalhes no capítulo 4.

Conversões tradicionais não atendem a necessidade deste trabalho devido à semântica específica da máquina de estados da aplicação. Transições relacionadas são sincronizadas em um único elemento *fork/join* para permitir a evolução de estados simultânea dos componentes. No elemento *fork/join* utilizado podem chegar e partir várias transições (no caso, as sincronizadas).

Esse aspecto requer um método específico que traduza as transições sincronizadas e relacionadas em *fork/join* na máquina de estados da aplicação para uma única transição com seu conjunto de arcos na rede de Petri correspondente.

Além disso, as redes de Petri utilizadas nessa abordagem são do tipo ordinárias, não havendo necessidade de usar as redes de alto nível como as utilizadas nas pesquisas relacionadas.

A análise do comportamento dos componentes e da aplicação é feita considerando as propriedades das redes de Petri. O significado de cada propriedade no contexto dos componentes foi objeto de estudo

desse trabalho e será visto em detalhes no capítulo 4. A maior inovação dessa proposta é a identificação de possíveis problemas no sistema (além de *deadlock)* que podem ser ocasionados devido à conexão dos componentes, e também a comparação do comportamento do componente individualmente com o comportamento dele quando conectado a outros para formar uma aplicação.

Para a avaliação desta proposta seria necessário que toda a abordagem proposta por Silva (2009) pudesse ser testada e avaliada. Isso se mostrou totalmente viável dentro da nova versão do ambiente SEA e pelo suporte do *framework* OCEAN. Para tanto, ferramentas de análise estrutural e comportamental de componentes deveriam ser inseridas no ambiente.

3.2.3 Comparação da abordagem proposta com os trabalhos correlatos

A tabela 3.2 resume as principais características que diferenciam essa proposta das demais analisadas. As principais diferenças são:

- *A forma como os componentes são especificados*: essa dissertação utiliza a abordagem de Silva (2009), que sugere a especificação de componentes, e de software baseado em componentes, exclusivamente com UML, uma única linguagem. Outras propostas utilizam mais de uma linguagem, o que exige o domínio de diferentes linguagens pelo projetista.
- *Comportamento da aplicação*: para a análise comportamental, essa dissertação considera também o comportamento da aplicação. Isso inclui a união do comportamento de todos os componentes conectados. Algumas propostas consideram apenas a conexão entre dois componentes.
- *Identificação de outros possíveis problemas comportamentais*: a análise comportamental proposta nesse trabalho identifica além de *deadlock*, outros problemas que podem ser cruciais para a execução do sistema baseado em componentes. Esses problemas são classificados como “Erros” ou “Advertências”. Erros são bloqueios que podem ocorrer devido a restrições na ordem dos serviços dos componentes envolvidos ou a serviços, fornecidos ou requeridos, por portos não conectados, como veremos com mais detalhes no capítulo 4. Advertências são situações suspeitas que demandam a análise do usuário. Esses problemas são identificados e apresentados ao usuário por

meio de relatórios. Os trabalhos correlatos identificam apenas *deadlock* e não mencionam o motivo da ocorrência do mesmo.

- *Comparação do comportamento individual do componente com o comportamento dele após ser conectado a outros para formar uma aplicação*: o comportamento de um componente pode ser alterado quando ele é conectado a outros para formar uma aplicação. Serviços disponíveis podem ficar temporariamente disponíveis ou até indisponíveis quando o componente é conectado a outros, assim como comportamentos repetitivos podem deixar de sê-lo. Na análise comportamental essas alterações são identificadas e apresentadas ao usuário, que deve analisar se representam um problema ou não. Os trabalhos correlatos não tratam esse tipo de análise.

3.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Todos os trabalhos referidos contribuíram de alguma forma para a definição da solução proposta para a análise de compatibilidade de componentes sugerida nesse trabalho. Todos os trabalhos relacionados na seção 3.1 firmam a importância e a necessidade da análise de compatibilidade de componentes, em sistemas baseado em componentes, para que as aplicações construídas a partir desse paradigma não entrem em bloqueio.

Apesar das propostas utilizarem modelos distintos para a especificação da interface dos componentes, e de uma variedade de técnicas para realizar a análise de componentes, geralmente, em relação à compatibilidade comportamental, o único problema verificado é o bloqueio do sistema.

Observa-se que a proposta desta dissertação faz contribuições ainda não exploradas na literatura, como a identificação de outros possíveis problemas comportamentais - além de *deadlock* - e a comparação do comportamento do componente individual com o comportamento desse mesmo componente dentro da aplicação. Além disso, o motivo do bloqueio é identificado.

No próximo capítulo a abordagem utilizada é apresentada em detalhes, assim como os resultados das análises propostas.

Tabela 3.2: Comparação dos trabalhos correlatos com essa dissertação

Trabalho	**Utiliza uma única linguagem de especificação**	**Realiza análise estrutural**	**Realiza análise comportamental**	**Considera o comportamento do conjunto de componentes conectados**	**Identifica outros possíveis problemas comportamentais além de deadlock**	**Compara o comportamento do componente com o que ele apresenta quando conectado a outros**
DIAS e VIERIA 2000	X	Sim	Sim	Sim	X	X
PLASIL e VISNOVSKY 2002	X	X	Sim	X	X	X
MOUAKHER, LANOIX e SOUQUIÈRES, 2006	Sim	Sim	Sim	X	X	X
SILVA 2000	X	Sugere	Sugere	Sim	X	X
CUNHA 2005	X	Sim	Sim	Sim	X	X
SILVA 2009	Sim	Sugere	Sugere	Sim	X	X
Esta Dissertação	Sim	Sim	Sim	Sim	Sim	Sim

Legenda: X- Não Sugere- O trabalho não realiza a análise de compatibilidade mas indica a possibilidade de fazê-lo.

4 APRESENTAÇÃO DA ABORDAGEM PARA ESPECIFICAÇÃO E ANÁLISE DE COMPATIBILIDADE DE COMPONENTES

Este capítulo descreve a abordagem utilizada neste trabalho para a especificação do software orientado a componentes, que abrange a especificação da interface de componentes, considerando os aspectos estrutural e comportamental, e como eles são conectados, utilizando UML, dentro da nova versão do ambiente SEA.

Descreve também a técnica proposta neste trabalho para a análise de compatibilidade comportamental de componentes. Essa análise é realizada a partir da especificação das interfaces, a fim de detectar possíveis erros, como especificação inconsistente e incompatibilidades comportamentais entre os componentes da aplicação, além de situações passíveis de advertência, que devem ser analisadas pelo usuário.

Para a avaliação da proposta de análise comportamental, foi criada a ferramenta de análise comportamental (FAC) no ambiente SEA. Porém, a análise comportamental de componentes é realizada caso não haja problemas estruturais. Portanto, foi necessária também a criação da ferramenta de análise estrutural (FAE) no mesmo ambiente.

Todos os exemplos ilustrados neste capítulo foram criados e utilizados no desenvolvimento desse trabalho e toda a modelagem, feita no ambiente SEA.

A figura 4.1 ilustra o processo de especificação e análise de compatibilidde de componentes adotado neste trabalho. O processo é ilustrado por meio do diagrama de atividades. Todas as estapas, isto é, atividades e ações definidas no diagrama são descritas em detalhes nas próximas seções deste capítulo.

4.1 ESPECIFICAÇÃO ESTRUTURAL

A especificação estrutural diz respeito à relação das assinaturas de métodos da interface de componente.

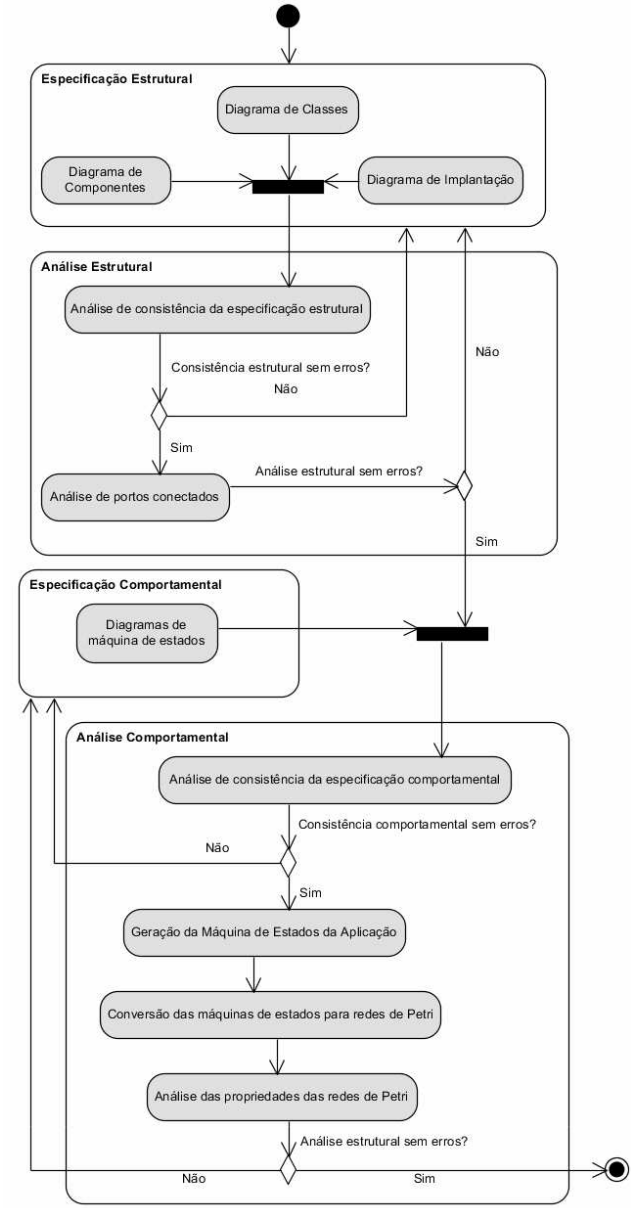


Figura 4.1: Processo de especificação e análise de componentes adotado neste trabalho.

4.1.1 Especificação estrutural da interface de componentes no ambiente SEA

> O diagrama de componentes é um dos diagramas da UML voltados a modelar software baseado em componentes. A finalidade do diagrama é especificar componentes e relacionamentos entre eles. Considerando a nomenclatura de UML, a interface de componente é composta por uma coleção de portos, cada um com especificações de interface (UML) associadas (SILVA, 2009, p.89).

Existem várias alternativas de representação de componente através do diagrama de componentes:

- Com ou sem especificação de interfaces;
- Com ou sem especificação de portos;
- Com ou sem associação de interfaces aos portos.

A escolha de uma ou outra depende do nível de detalhamento desejado na especificação. Porém, especificações que não consideram portos parecem ser mais limitadas, pois, nesse caso, supõe-se que o componente tenha apenas um porto. Logo, todas as possíveis interfaces do componente estarão conectadas a esse único porto, e qualquer outro componente que necessite se conectar a este, disporá de apenas um único canal de comunicação.

Na abordagem sugerida por Silva (2009) a especificação estrutural da interface de componente[7] demanda a especificação de todas as interfaces (no sentido de uma coleção de assinaturas de métodos) associadas ao componente em um ou mais diagramas de classes e a definição dos portos do componente, com a associação das interfaces requeridas ou fornecidas a cada um deles, em diagrama de componentes.

Portanto, para esse trabalho, faz-se necessário explicitar os portos dos componentes e as interfaces requeridas e fornecidas por cada porto, para que seja possível a realização das análises de compatibilidade.

No ambiente SEA a identificação de que a interface é requerida ou fornecida por um porto é feita através dos relacionamentos de "Realização" e "Dependência" entre porto e interface.

---

[7] A expressão "interface de componente" refere-se à parte externamente visível de um componente. Esse termo contrasta com a expressão "interface", de UML, que se refere a uma coleção de assinaturas de métodos.

A figura 4.2 ilustra o diagrama de componentes do ambiente SEA. O componente *ComponenteA* possui somente um porto, *porto1* e nele está associada por realização a interface *interfaceY*. Isso significa que o componente implementa os métodos declarados nessa interface e que um elemento externo pode invocá-los através do porto *porto1*. Ao mesmo porto está associada a interface *interfaceX,* por dependência. Isso significa que o componente *ComponenteA* invoca os métodos declarados na interface *interfaceX* do elemento externo que estiver conectado ao porto *porto1*.

Cada uma das interfaces relacionadas ao porto de um componente no diagrama de componentes deve ser declarada em um ou mais diagramas de classes. É a partir dessa declaração que será possível verificar quais métodos são invocados ou requeridos a partir de um porto de um componente. Essa restrição é essencial para a realização das análises de compatibilidade entre componentes conectados em uma aplicação, propostas por este trabalho. A figura 4.3 ilustra o diagrama de classes do ambiente SEA.

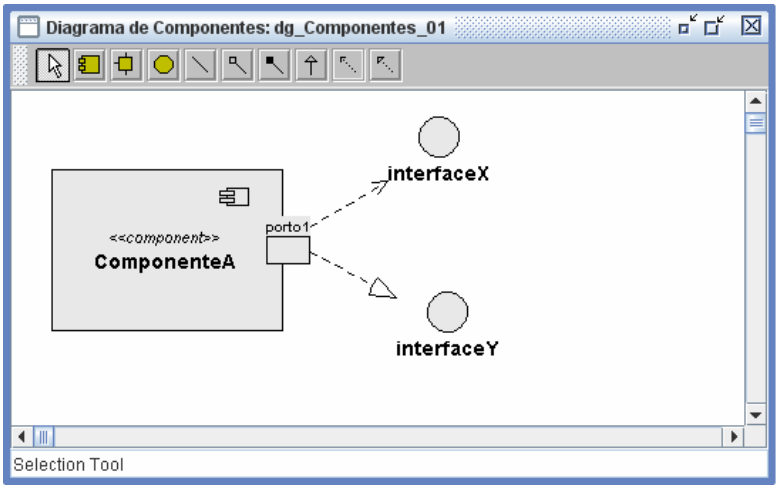


Figura 4.2: Diagrama de componentes do ambiente SEA.[8]

[8] As figuras 4.1 e 4.2 são *hard copy* de telas do ambiente SEA. Todos os demais diagramas referentes à especificação de componentes foram produzidos neste ambiente.

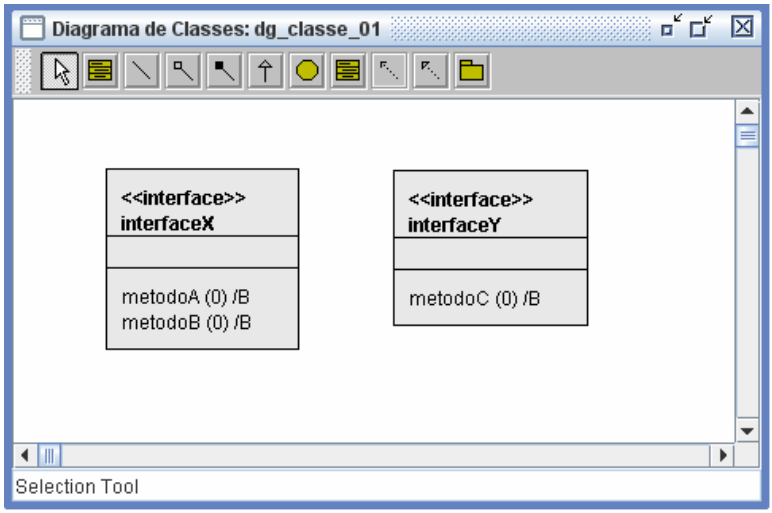


Figura 4.3: Diagrama de classes do ambiente SEA.

4.1.2 Especificação estrutural de um sistema baseado em componentes no ambiente SEA

No DSBC um artefato de software é composto pela conexão de dois ou mais componentes, através de seus portos e interfaces.

Conforme a abordagem sugerida por Silva (2009, p.193) o mínimo necessário para especificar um software baseado em componentes consiste na descrição:

- Do conjunto de componentes utilizado em um software, representado por meio de diagrama de componentes,
- Do conjunto de interfaces associadas aos portos, por meio de diagrama de classes;
- Da organização dos componentes, representada por meio do diagrama de implantação (*deployment diagram*).

Especificados quais componentes fazem parte de uma aplicação (em diagrama de componentes), é necessário definir como esses componentes são organizados para compor a aplicação. Nesse trabalho, isso é feito conforme a abordagem de Silva (2009, p.194), da seguinte forma:

- Em um diagrama de implantação, definir as instâncias de nodos necessárias para a aplicação;
- Definir os artefatos (instâncias de componentes) em cada instância de nodo;
- Estabelecer a interligação dos elementos.

A figura 4.4 ilustra um artefato de software descrito em diagrama de implantação e constituído da interligação dos componentes *ComponenteA*, *ComponenteB* e *ComponenteC*, que são definidos no diagrama de componentes, ilustrado na figura 4.5.

4.2 ANÁLISE ESTRUTURAL

> Para que dois componentes possam ser conectados, é necessário que suas interfaces sejam compatíveis, isto é, que independentemente de homogeneidade de linguagem em que estão implementados, de localização física e de plataforma de execução, os serviços requeridos por um, estejam disponíveis no outro e possam ser invocados, quando necessário. A descrição de um componente deve permitir verificar esta compatibilidade – senão, é impossível verificar se dois componentes podem ou não ser conectados (SILVA, 2009, p.177).

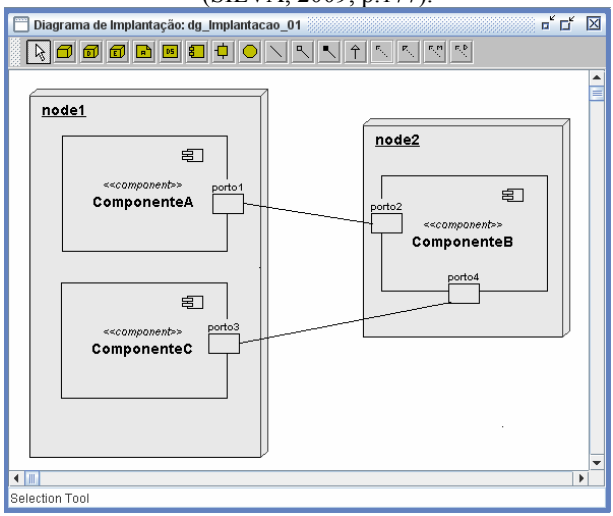


Figura 4.4: Artefato de software constituído da interligação de componentes.

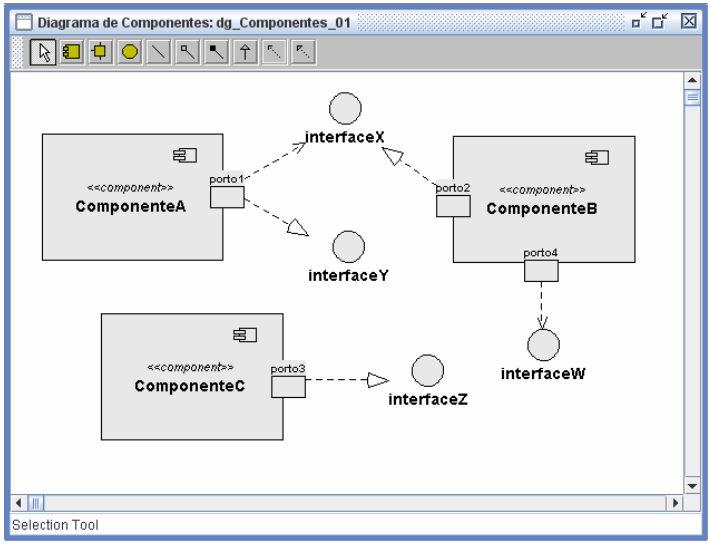


Figura 4.5: Declaração de componentes em diagrama de componentes.

A análise da compatibilidade estrutural é realizada para cada par de portos conectados dos diferentes componentes da aplicação. Métodos requeridos no porto em um lado da conexão devem ser fornecidos pelo porto do outro lado da conexão.

4.2.1 Incompatibilidade estrutural

A incompatibilidade estrutural pode ocorrer na conexão entre portos de componentes quando o serviço requerido por um componente não tem o respectivo serviço fornecido pelo outro. Isso pode ocorrer por diferentes motivos, como: inexistência do serviço, tipo de serviço divergente, quantidade de parâmetros divergentes e tipo de parâmetros divergentes - ou até algumas dessas possibilidades combinadas.

Considere a figura 4.4 e a figura 4.5. O componente *ComponenteA* tem associada a seu porto *porto1* a interface *interfaceX* por dependência, e o componente *ComponentB* tem associada a seu porto *porto2* a interface *interfaceX* por realização. Esses dois componentes são conectados por meio de seus portos, *porto1* e *porto2*. Os componentes *componenteA* e *componenteB* não terão incompatibilidade estrutural. Isso é garantido porque a descrição da

interfaceX no diagrama de classe dentro do ambiente SEA é única. Isso garante que os métodos requeridos pelo componente *ComponenteA* são fornecidos pelo componente *ComponenteB* na ligação dos portos *porto1* e *porto2*.

Porém, nem sempre os componentes conectados terão as mesmas interfaces, com o mesmo nome, associadas aos seus portos. O componente *ComponenteC* tem associado ao seu porto *porto3* por realização a interface *interfaceZ*, e o componente *ComponentB* tem associado a seu porto *porto4* a interface *interfaceW* por dependência. Esses dois componentes são conectados através de seus portos, *porto3* e *porto4*. Nesse caso, não há garantias que os serviços requeridos pelo componente *ComponenteB* serão fornecidos pelo componente *ComponenteC*. Por isso, a análise estrutural entre os componentes conectados em uma aplicação faz-se necessária para assegurar que há compatibilidade estrutural entre eles e que o sistema modelado não entrará em colapso.

Conforme explicado no item 4.1.1, na abordagem utilizada nesse trabalho, um porto é relacionado às suas interfaces através dos relacionamentos de dependência e realização. Assim, o conjunto de métodos requeridos por um porto é o conjunto de métodos de todas as interfaces associadas por dependência. Esses métodos devem ser fornecidos pelo porto do outro lado da conexão através do seu conjunto de métodos fornecidos, isto é, o conjunto de métodos de todas as interfaces associadas por realização.

4.2.2 Ferramenta de análise estrutural - FAE

A ferramenta de análise estrutural (FAE), implementada nesse trabalho e inserida no ambiente SEA tem como entrada a especificação estrutural da interface de componentes, que inclui:

- Pares de portos conectados de diferentes componentes, através do diagrama de implantação;
- As interfaces dos componentes interligados, especificadas através do diagrama de componentes;
- A descrição de cada uma das interfaces associadas aos portos dos componentes em um ou mais diagramas de classes.

Depois de criada a especificação estrutural da interface de componentes, a FAE pode ser executada a partir do menu do ambiente SEA. Seu propósito é realizar a análise estrutural de componentes que consiste na análise de consistência da especificação estrutural e na análise de portos conectados. A figura 4.6 ilustra o processo de análise estrutural realizada pela FAE no ambiente SEA.

Na análise estrutural de componentes dois tipos de relatórios são gerados. Um com os possíveis problemas de consistência de especificação encontrados, e outro com os resultados da análise dos portos conectados.

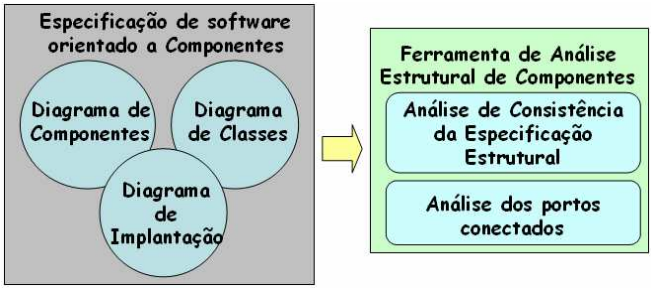


Figura 4.6: Ferramenta de análise estrutural – FAE.

4.2.2.1 Análise de consistência da especificação estrutural

A análise de consistência da especificação estrutural verifica se a especificação estrutural da interface de componentes obedece a todas as restrições necessárias, estabelecidas na abordagem. As restrições verificadas são listadas a seguir:

- Todos os componentes devem ser especificados no diagrama de componentes com ao menos um porto associado a ele. Componente sem porto associado denota especificação inconsistente;
- A cada porto, ao menos uma interface requerida ou fornecida deve ser associada. Porto sem interface denota especificação inconsistente;
- Cada interface definida no diagrama de componentes deve estar descrita em um diagrama de classes. Interfaces referenciadas em diagramas de componentes e não

descritas em diagrama de classes denotam especificação inconsistente;

- Interfaces definidas no diagrama de classes devem ter métodos declarados. Interfaces definidas em um diagrama de classes, mas sem métodos declarados, denotam especificação inconsistente;
- Deverá haver ao menos dois componentes conectados. Diagrama de implantação sem nenhuma conexão entre artefatos (instância de componentes) denota especificação inconsistente.

Ao final da análise é emitido o relatório de análise de consistência da especificação, listando os problemas encontrados na mesma. Um exemplo desse relatório é ilustrado na figura 4.7. Outros exemplos são vistos no capítulo 5 – Estudo de Caso.

4.2.2.2 Análise dos portos conectados

Caso a análise de consistência da especificação estrutural de componentes não detecte erros, a análise de portos conectados entre componentes é realizada. Essa análise compara, para cada par de portos conectados, os métodos requeridos em um porto com os métodos fornecidos pelo porto do outro lado da conexão, considerando nome do método, tipo de retorno, número de parâmetros e tipo de parâmetro. Os métodos são aqueles declarados nas interfaces associadas aos portos analisados.

Ao final da análise é emitido o relatório de análise de portos conectados, que identifica os problemas de compatibilidade estrutural. Um exemplo desse relatório é ilustrado na figura 4.8. Outros exemplos são vistos no capítulo 5 – Estudo de Caso.

```
************************************************************************
 Modelo analisado: dg_Componente_Exe01
 Modelo do tipo: Diagrama de Componentes
************************************************************************
------------------------------------------------------------------------
Componente: Comp01
------------------------------------------------------------------------
 Erro: Componente sem porto associado.
------------------------------------------------------------------------
Componente: Comp02
------------------------------------------------------------------------
 Componente analisado sem erros.
------------------------------------------------------------------------
Porto: porto01
------------------------------------------------------------------------
 Erro: Porto nao possui Interface associada.
------------------------------------------------------------------------
Porto: porto02
------------------------------------------------------------------------
 Porto analisado sem erros.
------------------------------------------------------------------------
Interface: interfaceX
------------------------------------------------------------------------
 Erro: Interface especificada em um Diagrama de Classes
 SEM metodos definidos.
------------------------------------------------------------------------
Interface: interfaceY
------------------------------------------------------------------------
 Erro: Interface nao especificada em um Diagrama de Classes
------------------------------------------------------------------------
Modelo com erros.
------------------------------------------------------------------------
************************************************************************
 Modelo analisado: dg_Implantação_Exe01
 Modelo do tipo: Diagrama de Deployment
************************************************************************
 Advertência: Porto [porto02] do componente [Comp02] sem conexão.
------------------------------------------------------------------------
Modelo com Advertência.
------------------------------------------------------------------------
```

Figura 4.7: Relatório de análise de consistência da especificação estrutural de componentes.

```
************************************************************
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Estrutural de Componentes
************************************************************
------------------------------------------------------------
 Analisando Componente [CompA] no porto [P1]
  conectado ao Componente [CompB] no porto [P2]
- Servico [metX(par:String): String].
    Fornecido como [metX(par1:Integer): String].
    Possui tipo de parametros diferentes.
- Servico [metK(): String].
    Servico NAO fornecido.
------------------------------------------------------------
 Analisando Componente [CompB] no porto [P2]
  conectado ao Componente [CompA] no porto [P1]
- Servico [metY(): String].
    Servico [metY] encontrado OK.
- Servico [metW(par1:String; par2: Integer): String].
    Servico [metW] encontrado OK.
------------------------------------------------------------
 Analisando Componente [CompC] no porto [P3]
  conectado ao Componente [CompB] no porto [P4]
- Servico [metZ(): Boolean].
    Servico [metZ] encontrado OK.
------------------------------------------------------------
 Analisando Componente [CompB] no porto [P4]
  conectado ao Componente [CompC] no porto [P3]
- Servico [metY(): String].
    Servico NAO fornecido.
- Servico [metW(par1:String; par2: Integer): String].
    Fornecido como [metW(par1:String): Integer].
    Possui tipo de retorno diferente.
    Quantidade de parametros diferentes.
------------------------------------------------------------
Analise Estrutural de Componentes com erros.
------------------------------------------------------------
Fim da Analise Estrutual.
------------------------------------------------------------
```

Figura 4.8: Relatório de análise estrutural de componentes.

4.2.2.3 Algoritmos utilizados

Em resumo, o algoritmo de análise de consistência da especificação estrutural de componentes é descrito na figura 4.9. O procedimento da análise de portos conectados segue o algoritmo da figura 4.10. Dentro dessa análise é realizada a análise dos métodos requeridos e fornecidos que segue o algoritmo descrito na figura 4.11.

```
1  Para todos os diagramas de componentes da especificação
2   Para todos os componentes do diagrama de componentes
3    Se não tem nenhum porto associado
4     Erro: Componente sem porto associado.
5    Fim-Se
6   Fim-Para
7   Para todos os portos do diagrama de componentes
8    Se não tem nenhuma interface associada através de realização ou dependência
9     Erro: Porto sem interface.
10   Fim-Se
11  Fim-para
12  Para todas as interfaces do diagrama de componentes
13   Se a interface não foi declarada em algum diagrama de classes da especificação
14    Erro: Interface não declarada em diagrama de classes.
15   Senão
16    Se a interface não possui métodos declarados
17     Erro: Interface sem métodos declarados.
18    Fim-Se
19   Fim-Se
20  Fim-para
21 Fim-Para
22 Para todos os diagramas de implantação da especificação
23  Se não tem pelo menos uma conexão entre instância de componentes
24   Erro: Diagrama sem componentes conectados.
25  Fim-Se
26  Para todos os portos associados a instancia de componentes
27   Se não tem conexão com outro porto
28    Advertencia: Porto sem Conexão.
29   Fim-Se
30  Fim-Para
31 Fim-Para
```

Figura 4.9: Algoritmo de análise de consistência da especificação estrutural de componentes.

4.3 ESPECIFICAÇÃO COMPORTAMENTAL.

A especificação comportamental diz respeito às restrições na ordem de invocação de métodos requeridos e fornecidos pelo componente.

4.3.1 Especificação comportamental de componentes no ambiente SEA

Neste trabalho, que adota a abordagem proposta por Silva (2009), a especificação comportamental de um componente é representada pelo diagrama de máquina de estados da UML.

> A idéia básica é que cada estado represente um conjunto de métodos fornecidos ou requeridos que podem ser invocados em um determinado instante. Cada transição que sai de um estado representa a execução de um método fornecido ou requerido que pode deixar o componente no mesmo estado (isto é, o mesmo conjunto de métodos pode ser executado novamente) ou levar a outro estado, caracterizado por outro conjunto

de métodos que pode ser executado (SILVA, 2009, p.185).

```
1  Para todos os diagramas de implantação da especificação
2   Para toda conexão porto1-porto2
3    Identificar interfaces associadas aos portos no diagrama de componentes
4    Se houver interface associada por dependência no porto1
5      Obter a lista de todos os métodos requeridos do porto1
6      Se houver interface associada por realização no porto2
7        Obter a lista de todos os métodos fornecidos do porto2
8        Analisar métodos requeridos e fornecidos
9      Senão
10       Erro Estrutural: Porto com interface requerida associado a outro sem
                        interfaces fornecidas.
11     Fim-se
12   Senão
13     Se não houver interface associada por dependência no porto2
14      Erro Estrutural: Conexão entre portos que possuem apenas interfaces fornecidas.
15     Fim-Se
16   Fim-Se
17   Se houver interface associada por dependência no porto2
18     Obter a lista de todos os métodos requeridos do porto2
19     Se houver interface associada por realização no porto1
20       Obter a lista de todos os métodos fornecidos do porto1
21       Analisar métodos requeridos e fornecidos
22     Senão
23       Erro Estrutural: Porto com interface requerida associado a outro sem
                        interfaces fornecidas.
24     Fim-se
25   Senão
26     Se não houver interface associada por dependência no porto1
27       Erro Estrutural: Conexão entre portos que possuem apenas interfaces fornecidas.
28     Fim-Se
29   Fim-Se
30  Fim-Para
31 Fim-Para
```

Figura 4.10: Algoritmo para análise de portos conectados entre componentes.

```
 1 Para todos os métodos requeridos
 2  Se o método existe na lista de métodos fornecidos
 3   Se tipo de retorno do método requerido divergente do método fornecido
 4    Erro Estrutural: Tipo de retorno divergente.
 5   Fim-Se
 6   Se quantidade de parâmetros do método requerido divergente do método fornecido
 7    Erro Estrutural: Quantidade de parâmetros divergente.
 8   Fim-Se
 9   Para cada parâmetro do método requerido e fornecido
10    Se tipo de parâmetro do método requerido divergente do método fornecido
11     Erro Estrutural: Tipo do parâmetro divergente.
12    Fim-Se
13   Fim-Para
14  Senao
15   Erro Estrutural: Método requerido não encontrado na lista de métodos fornecidos.
16  Fim-Se
17 Fim-Para
```

Figura 4.11: Algoritmo para análise de métodos requeridos e fornecidos.

Para a especificação de restrições comportamentais relacionadas a componentes, modeladas com diagramas de transição de estados (na UML2 chamado de máquina de estados) foram adotadas algumas convenções, estabelecidas por Silva (2009). São elas:

- Em relação aos estados:
  - Não associação de semântica ao estado final. Todo diagrama terá apenas um estado final. Com isso, o estado final também não terá identificador associado.
  - A omissão do estado final indica que de qualquer estado pode ocorrer transição para o estado final (omitido).
  - Os identificadores adotados para os demais estados são números inteiros ou combinações de letras e números. Servem apenas para diferenciá-los e não para descrever uma situação, pois quem estabelece a semântica da modelagem são as transições, rotuladas por métodos.

  (SILVA, 2009, p. 185)

- Em relação às Transições:
  - Transições podem ser rotuladas por métodos ou não.
  - Quando há pelo menos uma transição rotulada saindo de um estado, significa que apenas os métodos que rotulam as transições que saem podem ser executados naquele estado.
  - A inexistência de transição rotulada saindo de um estado significa que qualquer método do componente, requerido ou fornecido, pode ser executado.

  (SILVA, 2009, p. 185)

- Em relação à rotulagem de Transições:
  - As transições podem ser rotuladas apenas com assinaturas de métodos. Isso é viável quando não há possibilidade de má interpretação em termos de identificação do método e da situação de sua invocação, isto é, quando o método é fornecido e é acessível em um único porto ou não é relevante por qual porto é invocado. No caso geral, a convenção de rotulagem de transição adotada nessa abordagem é a seguinte:

> [<sentido>]
> [[<porto>’,’]*<porto>’.’][<método>’,’]*<método>
>
> Onde:
> - <sentido> é o estereótipo <<out>> para métodos invocados pelo componente e <<in>> para métodos fornecidos, sendo esse último opcional (a ausência do estereótipo denota método fornecido);
> - <porto> é o identificador do porto através do qual o método é invocado – quando omitido, significa que a invocação pode ocorrer através do único porto ou que a informação do porto não é relevante na modelagem em questão. Também é possível especificar mais de um porto;
> - <método> é o identificador do método invocado. Também é possível especificar mais de um método.
>
> (SILVA, 2009, p. 188)

No ambiente SEA, alguns conceitos relacionados a essa abordagem não existiam e foram criados para possibilitar a especificação de restrições comportamentais relacionadas a componentes.

Neste trabalho, cada componente especificado no diagrama de componentes deve ter seu comportamento modelado através de um diagrama de máquina de estados da UML, seguindo as convenções estabelecidas por Silva (2009) com algumas restrições, listadas a seguir:

- Caso a transição seja rotulada, ela deve ser rotulada com *<sentido>, <porto> e <método>;*
- Não será concebida a possibilidade de haver em uma transição mais de um *<porto>,* assim como mais de um *<método>;*

A figura 4.12 ilustra um exemplo de especificação comportamental de três componentes: *CompA*, *CompB* e *CompC*, por meio do diagrama de máquina de estados. *CompA*, em seu estado *E1*, invoca *metx()* através de seu porto *P1*, passando então para o estado *E2*. Nessa situação, *CompA* aguarda a invocação de *metY(),* fornecido através de seu porto *P1*. Ao receber essa invocação, *CompA* retorna ao estado *E1*.

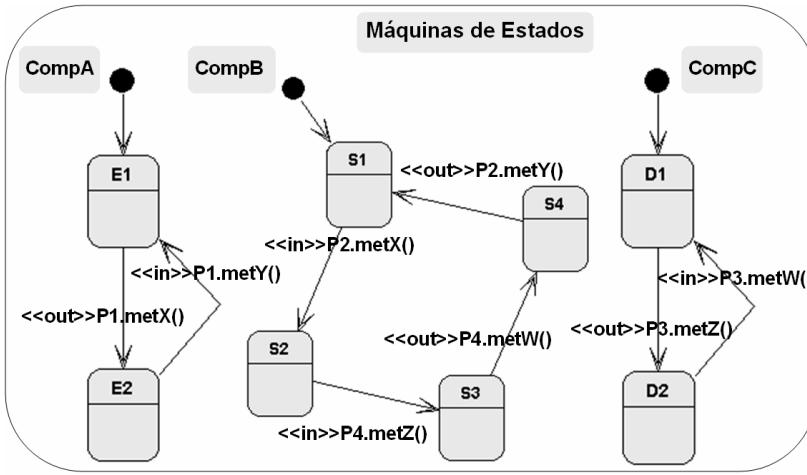


Figura 4.12: Especificação comportamental de componentes por meio do diagrama de máquina de estados.

4.3.2 Especificação comportamental de um sistema baseado em componentes no ambiente SEA

Uma vez especificados quais componentes fazem parte da aplicação (através do diagrama de componentes), especificado o comportamento de cada um deles (através do diagrama de máquina de estados) e como esses componentes são organizados (através do diagrama de implantação) podemos ter a especificação do comportamento da aplicação através da união das máquinas de estados de cada um dos componentes conectados. A união das máquinas de estados passa a compor uma máquina de estados única, que é a máquina de estados da aplicação. O método para a geração da máquina de estados da aplicação foi proposto por Silva (2009, p.199) e consiste em:

> 1. Identificar pares de transições relacionadas. Um par de transições é relacionado caso envolva portos interligados e a execução de um mesmo método, que é requerido de um lado e fornecido pelo outro;
> 2. Inserir pseudo-estados *fork* e *join* (um único elemento sintático) sincronizando as transições das diferentes máquinas. Esse vínculo converterá

o conjunto de máquinas de estado individuais em uma única, a máquina de estados da aplicação orientada a componentes;
3. Sincronizar as transições dos pseudo-estados iniciais das várias máquinas com pseudo-estados *fork* e *join*, como no caso anterior (preservando um único pseudo-estado inicial para a máquina resultante).

Durante essa pesquisa foram identificadas algumas particularidades que a máquina de estados da aplicação deve ter para possibilitar as análises necessárias relativas à compatibilidade comportamental de componentes. São elas:

- Os estados da máquina de estados da aplicação devem ter, além do identificador do estado, o identificador da instância do componente associado àquele estado. A máquina de estados da aplicação terá todos os estados de todos os componentes envolvidos no sistema, diferente das máquinas de estados dos componentes individuais, onde todos os estados são associados a um único componente. Essa identificação é necessária na análise comportamental para identificar quais são os estados não alcançáveis de um determinado componente.
- As transições da máquina de estados da aplicação devem ter, além de <<sentido>>, <<porto>> e <<método>>, o identificador da instância do componente. Esse identificador é necessário, na análise comportamental, para identificar componentes que podem ter métodos não executáveis ou executáveis somente algumas vezes, além de identificar componentes sem portos conectados, cujos métodos fornecidos são essenciais para seu funcionamento.

Essas particularidades são essenciais para identificar com precisão quais os componentes envolvidos nos problemas comportamentais encontrados na análise comportamental.

Dentro do ambiente SEA, a geração da especificação comportamental da aplicação é feita de forma automática, por meio de uma ferramenta implementada neste trabalho. Essa ferramenta é

executada na análise comportamental de componentes, como tratado na seção seguinte.

A figura 4.13 ilustra a máquina de estados da aplicação de um exemplo hipotético, gerada automaticamente no ambiente SEA. Cada transição possui o identificador da instância do componente em seu rótulo, além da identificação de <*sentido*>, <*porto*> e <*método*>.

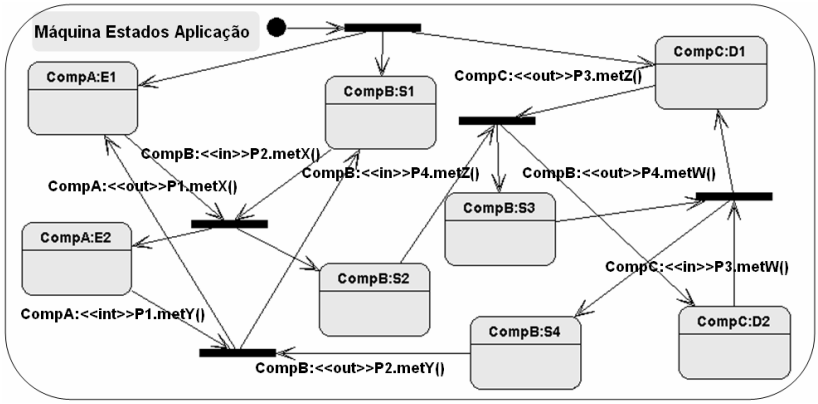


Figura 4.13: Especificação comportamental da aplicação orientada a componentes.

4.4 ANÁLISE COMPORTAMENTAL

> A análise da compatibilidade comportamental consiste em verificar se as restrições de ordem de invocação de métodos requeridos e fornecidos estabelecidas em cada componente são compatíveis com as dos demais componentes a ele interligados. Esse tipo de avaliação envolve todo o conjunto de componentes interligados – diferente da avaliação de compatibilidade estrutural, que trata um par de portos de cada vez. (SILVA, 2009, p.198).

Para a realização da análise de compatibilidade comportamental é necessário gerar a especificação comportamental da aplicação. Esse procedimento foi tratado na seção 4.3.2. Ela é representada pela máquina de estados da aplicação e é gerada através da interligação das

máquinas de estados individuais de cada componente envolvido, conforme proposto por Silva (2009), tendo algumas particularidades adicionadas por esta pesquisa.

4.4.1 Incompatibilidade Comportamental

Na literatura - os trabalhos pesquisados que tratam de análise de compatibilidade de componentes - a incompatibilidade comportamental está associada ao problema de *deadlock,* isto é, bloqueio do sistema (DIAS e VIEIRA, 2000), (SILVA, 2000), (PLASIL e VISNOVSKY, 2002), (CRAIG e ZUBEREK, 2004), (CHOUALI e SOUQUIÈRES, 2005), (CUNHA, 2005), (MOUAKHER, LANOIX e SOUQUIÈRES, 2006), (CRAIG e ZUBEREK, 2007) e (WEI e TONG, 2008). Considerando a abordagem utilizada nessa pesquisa, a questão é se a máquina de estados da aplicação consegue ter a evolução de estados esperada ou em algum momento fica bloqueada, sem possibilidade de proceder a novas transições.

Porém, neste trabalho, além da identificação de bloqueio no sistema, buscou-se identificar outros problemas relacionados à compatibilidade comportamental de componentes. Segundo Szyperski (2003) e Brereton (1999) mesmo conhecendo as características de um componente, não há garantia quanto ao resultado da combinação do mesmo com outros componentes no desenvolvimento de um sistema. É necessário tentar prever o maior número possível de problemas ocasionados por essa combinação. Além disso, quando identificado bloqueio no sistema, é possível identificar a causa do bloqueio. Essas questões são tratadas na seção 4.5.

4.4.2 Ferramenta de análise comportamental – FAC

Para realizar a análise comportamental de componentes neste trabalho, os diagramas de máquinas de estados dos componentes individuais e da aplicação são convertidos em redes de Petri correspondentes, como já explicado no capítulo 3, seção 3.2.2. O uso das redes de Petri foi motivado pela possibilidade de utilização de ferramentas de análise já existentes e consolidadas no mercado, além de pesquisas anteriores no mesmo laboratório de pesquisa deste trabalho que já utilizavam redes de Petri no contexto de componentes. Essa transformação é feita de forma totalmente transparente para o usuário,

que não necessita ter qualquer conhecimento sobre rede de Petri. Assim como a transformação, as análises também são realizadas de forma automatizada: o usuário recebe apenas o resultado final dessas análises, sem ver ou manipular redes de Petri.

Problemas comportamentais são identificados a partir da análise das propriedades das redes de Petri. A interpretação de cada propriedade para o contexto abordado foi objeto de estudo deste trabalho e está relatada na seção 4.5.

Alguns problemas, como a identificação de bloqueio no sistema, são analisados e relacionados como erro comportamental. Outros, porém, como métodos que nunca serão executados, são identificados, mas destacados como advertência. A advertência consiste na identificação de uma característica do sistema que pode ser considerada um erro comportamental ou não, pois depende da avaliação do usuário. Para que as advertências possam ser classificadas como erros comportamentais ou não, de forma automatizada, algumas informações adicionais sobre os componentes e a aplicação deveriam ser inseridas na especificação, como por exemplo, se o componente/aplicação deve ser reiniciável ou não; se um determinado serviço deve estar sempre disponível ou pode ficar temporariamente disponível ou até mesmo indisponível. Esse trabalho não tratou deste tipo de informação, mas faz indicações para que seja abordada em trabalhos futuros.

Depois de criada a especificação comportamental de todos os componentes e identificado que não há nenhum erro estrutural através da FAE, a ferramenta de análise comportamental (FAC) pode ser executada a partir do menu do ambiente SEA. Seu propósito é realizar a análise comportamental de componentes, que consiste na análise de consistência da especificação comportamental, geração da máquina de estados da aplicação, conversão das máquinas de estados em redes de Petri e análise das propriedades das redes de Petri. A figura 4.14 ilustra o processo de análise comportamental realizada pela FAC no ambiente SEA.

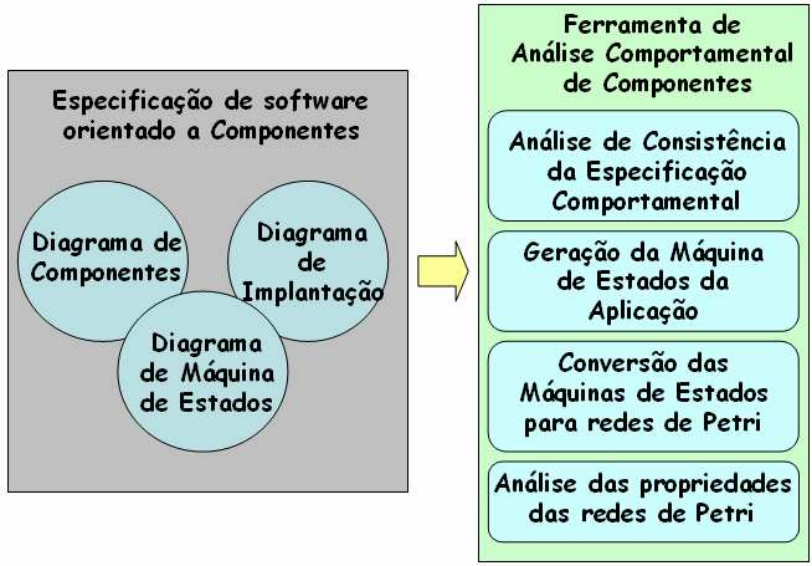


Figura 4.14: Ferramenta de análise comportamental – FAC.

4.4.2.1 Análise de consistência da especificação comportamental

Assim como a análise estrutural necessita primeiramente de uma análise de consistência da especificação estrutural de componentes, a análise comportamental também necessitará de uma análise de consistência da especificação comportamental. Essa análise envolve os diagramas de máquinas de estados definidos para cada componente e o diagrama de componentes. Essas verificações são necessárias para garantir que as restrições estabelecidas na abordagem para a especificação comportamental dos componentes foram cumpridas. Só assim, a máquina de estados da aplicação pode ser gerada. Caso haja erro na análise de consistência da especificação comportamental, a máquina de estados da aplicação não poderá ser gerada e a análise comportamental do sistema baseado em componentes não será realizada.

As restrições verificadas na análise de consistência da especificação comportamental dos componentes são:

- todo componente declarado no diagrama de componentes deve possuir uma máquina de estados associada;

Para cada máquina de estados que representa o comportamento de componente:

- Deve existir apenas um pseudo estado inicial;
- Deve existir um ou nenhum estado final;
- Deve existir pelo menos um estado (que não seja estado final);
- Deve existir pelo menos uma transição associada a cada estado definido;
- Todas as transições devem ser rotuladas com <<*sentido*>>, <<*porto*>> e <<*método*>>;

Algumas restrições são verificadas na edição do diagrama de máquina de estado:

- Os métodos que rotulam as transições devem ser especificados no diagrama de classes;
- Os portos que rotulam as transições devem ser associados ao componente;
- Quando o sentido da transição for <<*in*>>, o relacionamento do porto com a interface que contém o método deve ser de dependência;
- Quando o sentido for <<*out*>>, o relacionamento do porto com a interface que contém o método deve ser de realização.

Ao final da análise é emitido o relatório de análise de consistência da especificação comportamental, que analisa a máquina de estados de cada componente declarado. Um exemplo desse relatório é ilustrado na figura 4.15.

```
*****************************************************
 Modelo analisado: dg_maquina_CompC
 Modelo do tipo: Diagrama de Maquina de Estados
*****************************************************
-----------------------------------------------------
 Verificacao da existencia de Estado Inicial
-----------------------------------------------------
  Modelo com Estado Inicial definido ok.
-----------------------------------------------------
 Verificacao da existencia de Estado final
-----------------------------------------------------
  Advertência: Modelo sem definicao de Estado Final.
-----------------------------------------------------
 Estado: b1
-----------------------------------------------------
  Estado analisado sem erros.
-----------------------------------------------------
 Estado: b2
-----------------------------------------------------
  Estado analisado sem erros.
-----------------------------------------------------
Modelo sem erros.
-----------------------------------------------------
```

Figura 4.15: Relatório de análise de consistência da especificação comportamental de componentes.

4.4.2.2 Geração da máquina de estados da aplicação

Criada a especificação comportamental de todos os componentes envolvidos na aplicação, e constatado que não há erros de especificação comportamental por meio da análise de consistência comportamental, a máquina de estados da aplicação é gerada automaticamente, conforme descrito na seção 4.3.2.

A FAC executa a ferramenta de geração automática da especificação comportamental da aplicação e apresenta o diagrama de máquina de estados correspondente ao comportamento da aplicação na tela para o usuário, conforme exemplo da figura 4.13. Outros exemplos são vistos no capítulo 5 – Estudo de Caso.

4.4.2.3 Conversão das máquinas de estados para rede de Petri

A abordagem utilizada nesse trabalho sugere que o usuário manipule somente diagramas UML para especificar software orientado a

componentes. As máquinas de estados dos componentes individuais e da aplicação são convertidas automaticamente em redes de Petri correspondentes de uma maneira totalmente transparente para o usuário que não necessita ter qualquer conhecimento sobre esse tipo de modelo. No ambiente SEA, o usuário nunca se depara com diagramas de redes de Petri.

Conforme explicado no capítulo 3, conversões tradicionais utilizadas para converter modelos UML em redes de Petri não atendem as necessidades deste trabalho, devido à semântica específica da máquina de estados da aplicação, utilizada na abordagem. Por isso, foi criado um método específico para traduzir as transições sincronizadas e relacionadas em fork/join na máquina de estados da aplicação para uma única transição com seu conjunto de arcos na rede de Petri correspondente.

O algoritmo para conversão da máquina de estados para rede de Petri, criado e utilizado nesta pesquisa, é sumarizado nos seguintes passos:

1. Para cada estado da máquina de estados, criar um lugar na rede de Petri (pseudo estado inicial não é considerado).
2. Identificar os estados relacionados ao pseudo estado inicial e marcar os lugares correspondentes na rede de Petri com uma ficha (*token*).
3. Para cada conjunto de transições relacionadas com fork/join na máquina de estados da aplicação, criar uma transição com um conjunto de arcos, conectando-a aos seus lugares de entrada e saída. A figura 4.16 ilustra essa situação. As transições correspondentes ao método *met1*(), dos componentes *C1* e *C2*, conectados pelos portos *P1* e *P2*, são relacionas e estão sincronizadas em fork/join, sendo convertida para uma única transição com um conjunto de arcos na rede de Petri.
4. Para cada transição não relacionada à outra na máquina de estados, criar uma transição correspondente na rede de Petri e conectá-la com arcos aos seus lugares de entrada e saída (aplica-se às transições das máquinas de estados dos componentes individuais e às transições da máquina de estados da aplicação correspondentes a porto desconectado).
5. Para cada transição não relacionada à outra na máquina de estados da aplicação, inserir um lugar de entrada sem ficha e um lugar de saída e conectá-los à transição correspondente na rede de Petri.

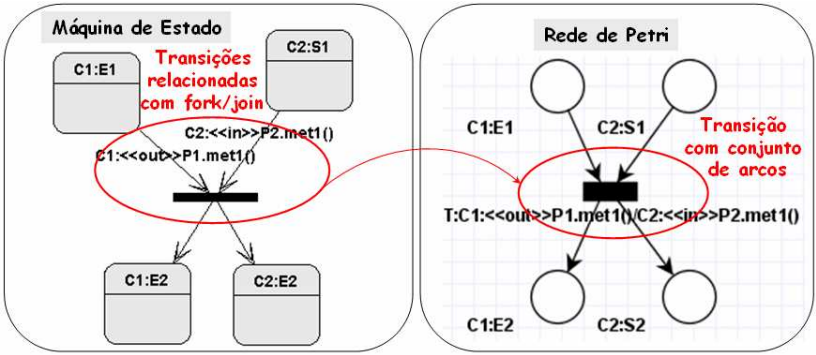


Figura 4.16: Exemplo de conversão da máquina de estados em rede de Petri.

A figura 4.17 ilustra de forma resumida o algoritmo para conversão das máquinas de estados para redes de Petri, descrito anteriormente.

As redes de Petri utilizadas neste trabalho são do tipo ordinária. A escolha por ordinária é devido à semântica que os lugares e fichas assumem no contexto dos componentes. Cada lugar representa um estado do componente, isto é, um conjunto de métodos fornecidos ou requeridos que podem ser invocados em um determinado instante, e as fichas representam condições lógicas que indicam se o componente está ou não em um determinado estado (lugar). Desta forma, não há necessidade de utilizar fichas de tipos particulares para o contexto utilizado.

4.4.2.4 Análise das propriedades das redes de Petri

Uma vez concluída a conversão das máquinas de estados - dos componentes individuais e da aplicação - em redes de Petri, é possível realizar análises comportamentais por meio da identificação das propriedades das redes de Petri.

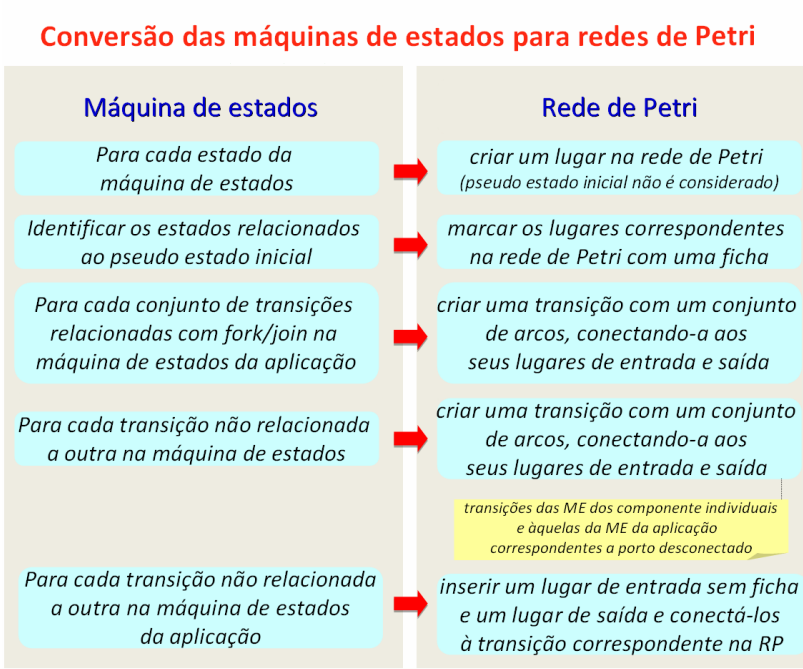


Figura 4.17: Algoritmo de conversão das máquinas de estados para redes de Petri.

O analisador da ferramenta Pipe - Platform Independent Petri net Editor 2, version 2.5 (PIPE, 2011) foi utilizado para identificar as propriedades das redes de Petri. O mesmo foi integrado ao ambiente SEA com adaptações. Essa integração foi possível devido ao código do analisador da ferramenta Pipe ser aberto. Essa característica motivou a escolha do mesmo. Dada uma rede de Petri, o analisador reporta a existência e características de uma dada propriedade. As propriedades verificadas pelo analisador da ferramenta Pipe são: binária, limitada, bloqueio e invariantes de transição. As demais propriedades: reiniciável, viva, transições quase vivas e transições mortas não são realizadas por esse analisador. Portanto, extensões desse analisador precisam ser realizadas dentro do ambiente SEA, para a análise dessas propriedades. Tais propriedades podem ser verificadas por meio da análise por enumeração das marcações, descrita no capítulo 2 seção 2.3.4, já que todas as redes de Petri geradas na abordagem são marcadas. Este trabalho não implementou essa extensão e sugere que seja feito em

trabalhos futuros. Para a geração do relatório de análise comportamental, essas informações são inseridas manualmente.

A interpretação de cada propriedade no contexto tratado foi objeto de estudo desse trabalho e está relatada na seção 4.5.

O relatório de análise comportamental é emitido ao final dessa análise, listando os problemas encontrados no software analisado. Alguns problemas são tratados como erros comportamentais e outros, como advertência. As advertências correspondem a situações que podem significar erro ou não: a conclusão depende da avaliação do usuário.

Um aspecto importante é a comparação de situações comportamentais que ocorrem no componente individualmente e no componente quando ele é conectado a outros para formar uma aplicação. Divergências comportamentais são identificadas e tratadas como advertências, ou seja, necessitam da análise do usuário.

Um exemplo desse tipo de relatório é ilustrado na figura 4.18. Outros exemplos são vistos no capítulo 5 – Estudo de Caso.

4.4.2.5 Algoritmos utilizados

Em resumo, o algoritmo para análise de consistência comportamental de componentes é descrito na figura 4.19. Caso as análises de consistência da especificação comportamental não retornem erro, é gerada a máquina de estados da aplicação, cujo algoritmo foi proposto por Silva (2009), descrito anteriormente na seção 4.3.2. Gerada a máquina de estados da aplicação, tanto ela como as máquinas de estados de cada componente individual serão transformadas em uma rede de Petri correspondente. Em resumo, o algoritmo para transformação da máquina de estados em rede de Petri dentro do ambiente SEA foi descrito na seção 4.4.2.3. Gerada as redes de Petri, a análise das propriedades é realizada, conforme o algoritmo descrito resumidamente na seção 4.5.2, que interpreta cada propriedade como veremos a seguir.

```
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Comportamental
--------------------------------------------------------------------------------
 Comportamento analisado: Componente [CompP]
--------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C1: metg(), metb()
  C2: metr(), metv()
--------------------------------------------------------------------------------
 Comportamento analisado: Componente [CompQ]
--------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C3: metq(), metp()
  C4: metr(), metv()
--------------------------------------------------------------------------------
 Comportamento analisado: Componente [CompL]
--------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [metg] fornecido pelo porto [P2].
- Ciclos de execucao de servicos
  C5: metb(), metq(), metp()
--------------------------------------------------------------------------------
 Comportamento analisado: Aplicação
--------------------------------------------------------------------------------
- Advertência: Aplicação não reiniciável.
- Advertência: aplicação com serviços temporariamente disponíveis.
  Serviços: [metb] requerido pelo componente [CompP] no porto [P1] e
                   fornecido pelo componente [CompL] no porto [P2].
            [metg] requerido pelo componente [CompP] no porto [P1] e
                   fornecido pelo componente [CompL] no porto [P2].
            [metq] requerido pelo componente [CompL] no porto [P4] e
                   fornecido pelo componente [CompQ] no porto [P3].
            [metp] requerido pelo componente [CompL] no porto [P4] e
                   fornecido pelo componente [CompQ] no porto [P3].
- Ciclos de execucao de serviços
  C6: metr(), metv()
--------------------------------------------------------------------------------
 Quadro comparativo dos serviços dos Componentes
--------------------------------------------------------------------------------
Met/Comp|    CompP      |     CompQ    |      CompL      | Aplicação
--------------------------------------------------------------------------------
  metg()|disponível(C1) |-             | temp.disponivel |temp.disponivel
  metb()|disponível(C1) |-             | disponível(C5)  |temp.disponivel
  metr()|disponível(C2) |disponível(C4)| -               |disponível(C6)
  metv()|disponível(C2) |disponível(C4)| -               |disponível(C6)
  metq()|-              |disponível(C3)| disponível(C5)  |temp.disponivel
  metp()|-              |disponível(C3)| disponível(C5)  |temp.disponivel
--------------------------------------------------------------------------------
Análise Comportamental com Advertências
--------------------------------------------------------------------------------
```

Figura 4.18: Relatório de análise comportamental de componentes

```
1   Para todos os diagramas de componentes da especificação
2    Para todos os componentes declarados
3     Se houver diagrama de máquina de estados para o componente
4      Se houver pelo menos um pseudo estado inicial
5        Se houver mais de um pseudo estado inicial
6          Erro: Mais de um pseudo estado inicial especificado
7        Fim-Se
8      Senão
9          Erro: Nenhum pseudo estado inicial especificado.
10     Fim-Se
11     Se houver pelo menos um estado declarado
12      Para todo estado do diagrama de máquina de estados
13       Se não houver transição associada ao estado
14        Erro: Estado sem transição associada.
15       Fim-Se
16      Fim-Para
17     Senão
18      Erro: Nenhum estado declarado.
19     Fim-Se
20     Para toda transição do diagrama de máquina de estados.
21      Se houver transição rotulada sem o trio sentido/porto/método
22       Erro: Transição rotulada inconsistente.
23      Fim-Se
24     Fim-Para
25    Senão
26     Erro: Componente sem comportamento especificado.
27    Fim-Se
28   Fim-Para
29  Fim-Para
```

Figura 4.19: Algoritmo de análise de consistência comportamental de componentes.

4.5 INTERPRETAÇÃO DAS PROPRIEDADES DAS REDES DE PETRI NO CONTEXTO DE COMPONENTES

"Interpretar uma rede de Petri implica antes de tudo em dar um sentido concreto a um modelo matemático, associando aos lugares, transições e fichas, os elementos existentes no sistema" (CARDOSO e VALETTE, 1997, p.105). Portanto, uma mesma rede pode ser interpretada de diferentes maneiras, caso ela seja utilizada para representar diferentes contextos.

Na abordagem proposta, que trata de componentes de software, os lugares representam os possíveis estados dos componentes, ou seja, o conjunto de métodos fornecidos ou requeridos que pode ser invocado em um determinado instante. As transições estão associadas aos métodos fornecidos ou requeridos por eles, e que ao serem executados podem deixar os componentes no mesmo estado ou não. A execução de

um método é representada pelo disparo de uma transição. A presença de ficha nos lugares indica o estado em que os componentes se encontram.

Na rede de Petri que representa o comportamento da aplicação, os lugares representam estados dos diversos componentes conectados. Cada componente pode assumir um determinado estado em um dado instante. A marcação de qual estado se encontra cada componente, é que determina o estado da aplicação.

As redes de Petri geradas a partir das máquinas de estados, que representam o comportamento de um componente ou aplicação, são analisadas levando em consideração suas propriedades. A partir delas são feitas algumas considerações sobre a especificação da aplicação baseada em componentes. A análise tem como objetivo identificar falhas na especificação dos componentes envolvidos, incompatibilidade comportamental entre eles, ou advertências, quando encontradas características na aplicação que podem representar algum problema. Neste caso, o usuário pode decidir se ações corretivas devem ser executadas em relação à especificação.

A análise das propriedades das redes de Petri individuais é necessária para comparar situações que ocorrem no comportamento individual do componente, mas que deixam de ocorrer na aplicação, quando o componente é conectado a outros. No relatório de análise comportamental é exibido um quadro comparativo com as possíveis alterações de comportamento de um componente que podem ocorrer quando ele passa a ser parte de uma conexão dos componentes.

Com isso, espera-se ter uma especificação de sistema de software baseado em componentes mais precisa, menos propensa a falhas e com mais qualidade.

4.5.1 Propriedades analisadas

As análises comportamentais são realizadas considerando as propriedades das redes de Petri. Cada propriedade foi descrita no capítulo 2, seção 2.3.4. A seguir é dada a interpretação de cada propriedade no contexto dos componentes.

- **Binária, Salva ou Segura (*safe*)**

Em se tratando de componentes, os lugares representam o estado dos componentes. A rede de Petri que representa o comportamento da aplicação baseada em componentes deve ser binária, porque uma ficha em um lugar representa uma condição lógica, indicando o estado em que o artefato se encontra. A presença de mais de uma ficha em um lugar

significa erro comportamental e é exibido no relatório de análise comportamental como “Erro! Máquina de estados Inconsistente”.

O mesmo tratamento é dado para as redes de Petri que representam o comportamento dos componentes individual.

- **Limitada**

Uma rede salva é necessariamente limitada. Logo, se a rede que representa o comportamento da aplicação baseada em componentes for salva, ela também será limitada. Porém, é possível que uma rede seja limitada, mas não salva. Nesse caso, já será identificado um erro comportamental (máquina de estados inconsistente), pelo fato da rede não ser salva. Todas as redes não limitadas serão não salvas. Portanto a análise dessa propriedade, para este trabalho, é dispensada devido à análise da propriedade anterior que a abrange.

O mesmo, obviamente, ocorre para as redes de Petri que representam o comportamento dos componentes individuais.

- **Reiniciável**

A rede de Petri que representa o comportamento da aplicação baseada em componentes e que fornece serviços indefinidamente pode não retornar ao seu ponto inicial. Nesse caso, a aplicação após ser iniciada continua a operar, mas não retorna ao seu estado inicial.

Portanto a análise dessa propriedade por si só não permite que seja identificado um erro comportamental, pois não há informação na modelagem UML mostrando se a aplicação descrita deve ou não voltar ao seu estado inicial.

O mesmo significado é dado para as redes de Petri que representam o comportamento dos componentes individuais.

Neste trabalho, a conclusão de que a rede de Petri não é reiniciável causa uma advertência, identificada no relatório de análise comportamental como "Advertência! Componente/Aplicação não reiniciável". A situação deverá ser analisada pelo usuário.

Uma forma de permitir a análise dessa propriedade de forma totalmente automática, a fim de indicar se há erro ou não na especificação comportamental dos componentes ou da aplicação, seria fornecer, através do ambiente, subsídios aos usuários para que fossem informadas características adicionais do sistema, como reiniciável (*sim ou não*). Isso poderia ser parametrizado, de forma que o ambiente

entendesse tais informações e com base nelas pudesse melhorar ainda mais a análise comportamental. Este trabalho não tratou de tais extensões.

- **Bloqueio**

Uma rede de Petri com bloqueio representando o comportamento de uma aplicação baseada em componentes indica que todos os componentes da aplicação, em algum ponto, ficam impedidos de invocar métodos requeridos e, consequentemente, não recebem chamada de métodos fornecidos. Nesse caso, não há evolução na rede, o que caracteriza incompatibilidade comportamental dos componentes conectados.

Isso pode ocorrer por duas razões: a primeira razão ocorre porque as restrições associadas à ordem de execução dos métodos estabelecidas por um componente não são respeitadas pelo outro. Esse caso é identificado no relatório de análise como: “Erro! Aplicação bloqueada: problemas na ordem de execução dos métodos.”.

A segunda razão é devido a portos não conectados em um ou mais componentes da aplicação. O problema de portos não conectados, que podem causar bloqueio na rede da aplicação, é quando o componente requer ou fornece métodos, através desse porto, que são essenciais para o seu funcionamento. Para métodos fornecidos, em algum momento, o componente poderá esperar pela chamada de um método, mas essa chamada nunca ocorrerá e o componente ficará em estado de espera, bloqueando a aplicação. Para métodos requeridos, em algum momento, o componente precisará invocar uma operação, e ela não estará disponível.

Métodos requeridos e fornecidos em portos não conectados não denotam erro estrutural, pois essa análise considera apenas pares de portos conectados, e caso eles sejam essenciais para o funcionamento do componente, isso só será identificado na análise comportamental. Exemplos com esse tipo de problema serão vistos no Capítulo 5 – Estudo de Caso.

Na abordagem proposta, para que o problema de portos não conectados seja identificado, ao converter a máquina de estados da aplicação para redes de Petri, a FAC verifica se existem transições que deveriam estar relacionadas e não estão. Se sim, lugares de entrada sem fichas são criados para essas transições, fazendo com que elas nunca fiquem habilitadas. Caso elas sejam essenciais para a execução do sistema, será gerado automaticamente bloqueio na rede. A identificação

de transições que deveriam estar relacionadas e não estão são verificadas a partir dos rótulos das transições da máquina de estados.

No relatório de análise comportamental essa situação é identificada como: “Erro! Aplicação bloqueada devido a portos não conectados: <lista de componentes, portos e serviços>”.

Para as redes de Petri que representam o comportamento de um componente individual, caso elas apresentem bloqueio, não faz sentido pensar em incompatibilidade comportamental. Esse caso indica que o componente, em algum momento, não consegue executar nenhum método e a interpretação que é dada é de especificação inconsistente. Logo, é necessário que, se um componente for conectado a outro, a rede que representa seu comportamento não seja bloqueada.

No relatório de análise, para cada componente analisado, essa situação é identificada como: “Erro! Especificação comportamental inconsistente”.

Ainda assim, componentes conectados em uma aplicação, que tenham seus comportamentos representados através de rede de Petri não bloqueadas, podem produzir uma aplicação com bloqueio.

- **Viva**

Uma rede de Petri que representa a aplicação baseada em componentes e que possua a propriedade viva significa que todos os serviços dos componentes envolvidos são disponíveis. Isso significa que os métodos podem sempre executar a partir de uma sequência de execução de outros métodos, e que todos os estados dos componentes envolvidos são alcançáveis, o que caracteriza uma especificação sem erros. Logo, no contexto de componentes, esta é uma boa propriedade, pois elimina a possibilidade de bloqueio (que denota erros) e de métodos que nunca serão executados (visto mais adiante). Nesse caso, a mensagem mostrada no relatório é: “Componente/Aplicação especificação ok”.

O mesmo significado é dado para as redes de Petri que representam o comportamento dos componentes individual.

Contudo, a ausência dessa propriedade não denota necessariamente um erro comportamental. Uma rede de Petri não viva pode ter transições quase vivas ou mortas. O significado delas é visto a seguir.

- **Análise das transições quase vivas**

Uma transição quase viva representa um método temporariamente disponível, isto é, ele pode ser executado uma ou mais vezes, mas em algum momento ele não poderá ser mais executado. Essa característica leva a uma advertência, pois é necessário que o usuário avalie se a indisponibilidade de um método, em um certo momento da execução é um problema de compatibilidade comportamental. No relatório de análise é exibido "Advertência! Serviços do componente/aplicação temporariamente disponíveis: <Lista de serviços>.

O mesmo significado é dado para as redes de Petri que representam o comportamento dos componentes individuais.

- **Análise das transições mortas**

Uma transição morta representa um serviço indisponível, isto é, que nunca será executado. A análise desse aspecto também requer a ação do usuário para estabelecer se há ou não um problema de compatibilidade comportamental, ou seja, se a indisponibilidade permanente de um serviço é um problema para aplicação. É possível uma aplicação ter serviços não essenciais. Portanto, esse aspecto é também tratado como advertência, identificado no relatório de análise como: "Advertência! Componente/aplicação com serviços não disponíveis: <Lista de serviços>".

O mesmo significado é dado para as redes de Petri que representam o comportamento dos componentes individuais, se bem que nesse caso não faz muito sentido prever um serviço permanentemente indisponível. Se não denota erro (porque não impede a operação do componente), certamente denota baixa qualidade do projeto.

Assim como acontece com outras propriedades, características do sistema poderiam ser informadas pelo usuário possibilitando uma análise totalmente automatizada. Por exemplo: o usuário poderia informar quais os métodos que deveriam ser obrigatoriamente executados. Se algum deles estiver representado como uma transição morta na rede, configurar-se-ia um erro.

- **Análise dos Invariantes de Transição.**

O invariante de transição corresponde a uma sequência cíclica de serviços que pode ser repetida indefinidamente. Eles são

identificados para cada componente e para a aplicação e no relatório de análise são exibidos como: “Ciclos de execução de serviços: <lista de serviços>”. A tabela 4.5 apresenta um resumo da interpretação das propriedades das redes de Petri no contexto dos componentes.

4.5.2 Tabela comparativa dos serviços dos componentes

Após a análise das redes de Petri dos componentes individuais e da aplicação (isso envolve todas as propriedades citadas no item 4.5.1), os invariantes de transição da aplicação são comparados com os invariantes de transição das redes de Petri dos componentes individuais, porque possíveis ciclos de execução de serviços de um componente podem não ser possíveis quando ele é conectado a outros. Essa situação também origina uma advertência e requer a avaliação do usuário para definir se é um problema de compatibilidade comportamental ou não. No relatório de análise comportamental é exibida uma tabela com essa comparação: “Tabela comparativa dos serviços de componentes”.

Nessa tabela, as colunas representam os componentes e a última coluna, a aplicação, e as linhas, os serviços existentes. É informado se o serviço é disponível, indisponível ou temporariamente disponível em cada componente e na aplicação. Assim, é possível comparar a mudança de comportamento nos componentes conectados.

Em resumo, o algoritmo que analisa as propriedades das redes de Petri no contexto dos componentes é descrito na figura 4.20.

Na análise comportamental de componentes dois tipos de relatórios são gerados. Um para cada modelo de máquina de estados de componente analisado, com os possíveis erros de inconsistências na especificação encontrados, e outro com as análises comportamentais de componentes, considerando as propriedades das redes de Petri. Exemplos de cada tipo de relatório são mostrados no capítulo 5 – Estudo de Caso.

Tabela 4.1 : Resumo da interpretação das propriedades das redes de Petri

Propriedade	Mensagem	
	RP da Aplicação	**RP dos componentes individuais**
Binária	-	-
Não Binária	Erro! Máquina de estados Inconsistente	Erro! Máquina de estados Inconsistente
Limitada	-	-
Reiniciável	-	-
Não Reiniciável	Advertência! Componente/Aplicação não reiniciável	Advertência! Componente/Aplicação não reiniciável
Bloqueio	Erro! Aplicação bloqueada: problema na ordem de execução dos métodos.	Erro! Especificação comportamental do componente inconsistente
	Erro! Aplicação bloqueada devido a portos não conectados: < Lista de portos e serviços>	
Não Bloqueio	-	-
Viva	Componente/Aplicação com especificação ok.	Componente/Aplicação com especificação ok.
Transições quase vivas	Advertência! Componente /Aplicação com serviços temporariamente disponíveis: <Lista de serviços>	Advertência! Componente /Aplicação com serviços temporariamente disponíveis: <Lista de serviços>
Transições mortas	Advertência! Componente/Aplicação com serviços não disponíveis: <Lista de serviços>	Advertência! Componente/Aplicação com serviços não disponíveis: <Lista de serviços>
Invariante de Transição	Ciclos de execução de serviços: <Lista de serviços>	Ciclos de execução de serviços: <Lista de serviços>

```
1  Para toda rede de Petri
2   Se rede não Salva
3    Erro: O componente pode assumir dois estados ao mesmo tempo.
4   Fim-Se
5   Se rede não Reiniciável
6    Advertência: O componente/aplicação retorna ao seu estado inicial.
7   Fim-Se
8   Se rede Bloqueada
9    Se portos não conectados (verificada apenas para a rede de Petri da aplicação)
10    Erro: Aplicação Bloqueada devido a portos não conectados.
11   Senao
12    Erro: Componente/Aplicação apresenta bloqueio.
13   Fim-Se
14  Senão
15   Se rede não Viva
16    Se há transições quase vivas
17     Advertência: Componente/Aplicação com métodos temporariamento disponíveis.
18    Fim-Se
19    Se há transições mortas
20     Advertência:Componente/Aplicação com métodos indisponíveis.
21    Fim-Se
22   Fim-Se
23  Fim-Se
24  Se há invariantes de transição
25   Listar métodos com sequencia cíclica.
26  Fim-Se
27 Fim-Para
```

Figura 4.20: Algoritmo para análise das propriedades das redes de Petri.

4.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Esse capítulo apresentou a abordagem utilizada nesse trabalho para especificação de software orientado a componentes, em UML.

De forma detalhada, explicou-se como proceder para produzir a especificação estrutural da interface de componentes no ambiente SEA. Tendo essa especificação, a análise estrutural pode ser realizada automaticamente através da ferramenta de análise estrutural (FAE) implementada no ambiente. O processo de análise executado pela FAE foi descrito detalhadamente.

Posteriormente, foi apresentado como produzir a especificação comportamental de cada componente através do diagrama de máquina de estados. Tendo a especificação comportamental de todos os componentes conectados, a especificação comportamental da aplicação pode ser gerada automaticamente através da implementação de um método que une todas as máquinas de estados dos componentes individuais. Depois de criada a especificação comportamental dos componentes individuais e verificado que não há erros estruturais, a análise comportamental pode ser executada através da ferramenta de análise comportamental (FAC) implementada no ambiente. O processo de análise executado pela FAC foi descrito detalhadamente.

Por fim, foi explicado como cada propriedade é interpretada no contexto dos componentes.

5 ESTUDO DE CASO

Esse capítulo apresenta estudos de caso, ilustrados com exemplos, com o objetivo de demonstrar a abordagem proposta.

Foram considerados dois sistemas: um sistema de reserva de hotel - uma variante do estudo de caso usado por Chouali, Souquières e Heisel (2006) – e um sistema de jogo de tabuleiro. A escolha desses exemplos foi motivada por serem sistemas cujos domínios são amplamente conhecidos. Acredita-se que essa característica facilita o entendimento do leitor.

O objetivo desses estudos de caso é ilustrar os possíveis problemas que podem ocorrer na conexão de componentes. Algumas incoerências e incompatibilidades foram propositalmente inseridas para avaliar as análises realizadas. Todas as situações analisadas são apresentadas a seguir.

5.1 CRIANDO A ESTRUTURA DOS COMPONENTES NO AMBIENTE SEA

Ao criar uma especificação UML2 no ambiente SEA, um conjunto de modelos ou diagramas pode ser criado a partir do menu opção “Arquivo->Novo modelo”. A figura 5.1 exibe a tela inicial da especificação de sistemas no ambiente SEA.

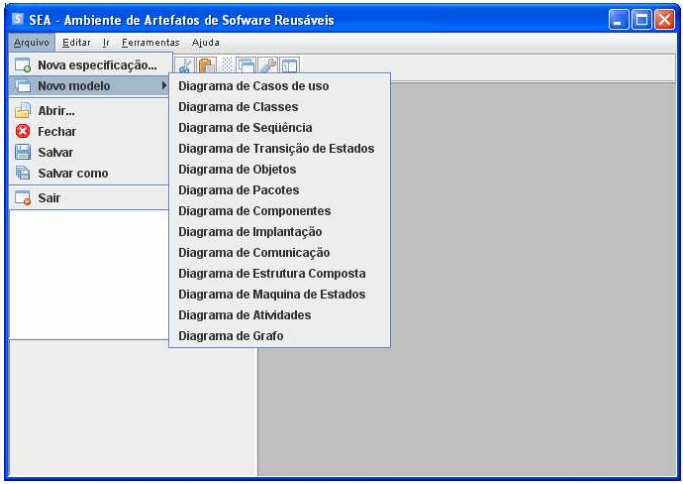


Figura 5.1: Tela Inicial do ambiente SEA.

5.1.1 Definindo componentes e suas conexões

Neste estudo o primeiro modelo criado no ambiente para o desenvolvimento orientado a componentes é o diagrama de componentes, para declarar todos os componentes envolvidos no sistema. Vamos considerar um sistema de reserva de hotel. Inicialmente, o sistema possui dois requisitos: solicitar reserva e confirmar reserva[9]. Após o cliente - que também pode ser um funcionário - solicitar a reserva para o hotel, o mesmo registra a solicitação e retorna a confirmação para o cliente. O sistema é modelado com três componentes: *Cliente*, *Hotel* e *Banco_Dados*. A figura 5.2 ilustra o diagrama de componentes inicial com os componentes declarados.

O próximo modelo que vamos criar é o diagrama de classes para definir as interfaces declaradas no diagrama de componentes. A figura 5.3 exibe o diagrama de classes com as interfaces definidas. Por último, a conexão dos componentes envolvidos é definida no diagrama de implantação, exibido na figura 5.4, que define o sistema baseado em componentes.

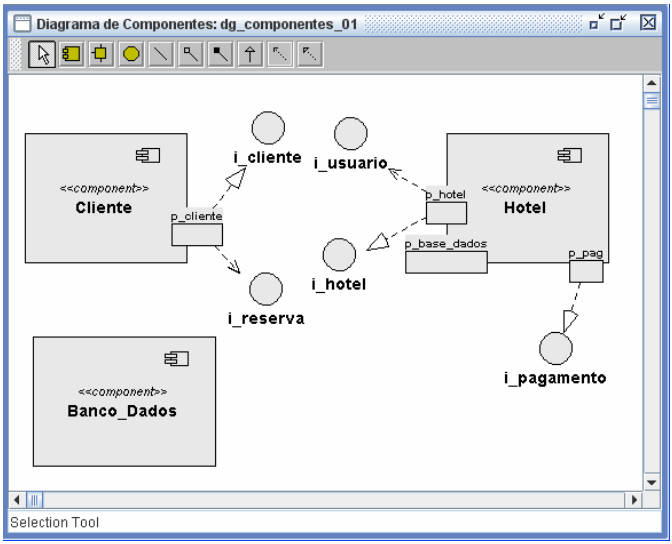


Figura 5.2: Diagrama de componentes inicial do sistema de reserva de hotel.

[9] Ao longo do capítulo, outros requsitos funcionais são inseridos e discutidos.

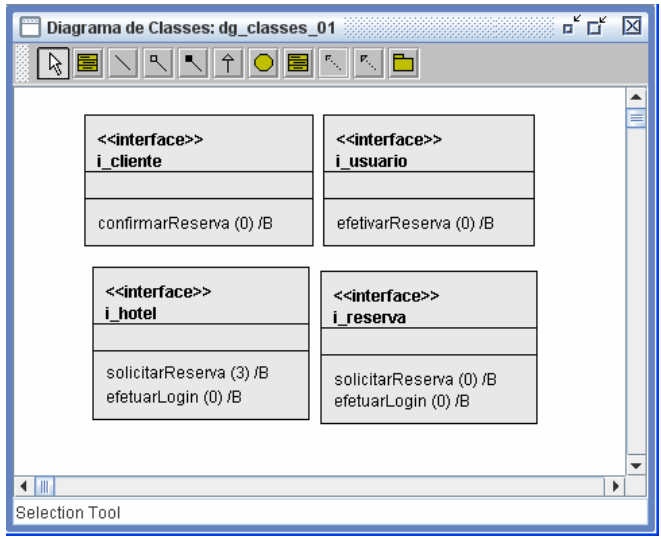


Figura 5.3: Diagrama de classes com parte das interfaces definidas.

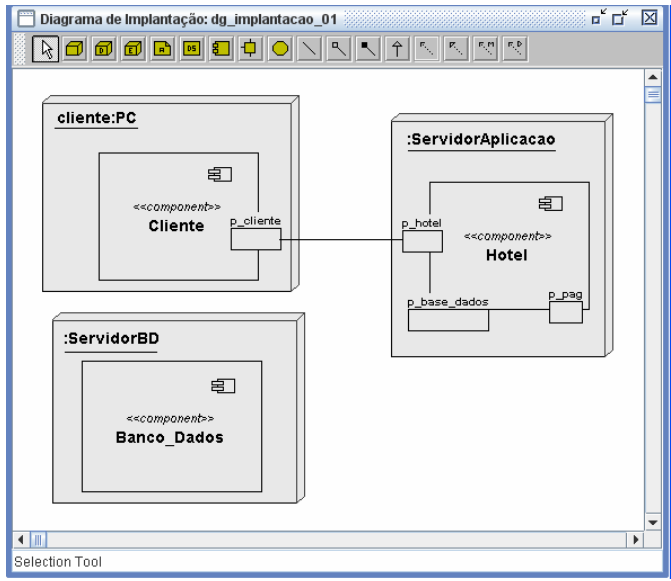


Figura 5.4: Diagrama de Implantação com dois componentes conectados.

5.2 REALIZANDO A ANÁLISE ESTRUTURAL DE COMPONENTES NO AMBIENTE SEA

Criada a especificação estrutural de componentes, que inclui a declaração dos componentes em diagrama de componentes, interfaces definidas em diagrama de classes e como os componentes são conectados em diagrama de implantação, a análise estrutural de componentes pode ser realizada.

Para realizar essa análise, é necessário acessar a opção do menu "*Ferramentas -> Component Structural Analyser*". A figura 5.5 ilustra a ferramenta de análise estrutural sendo acionada.

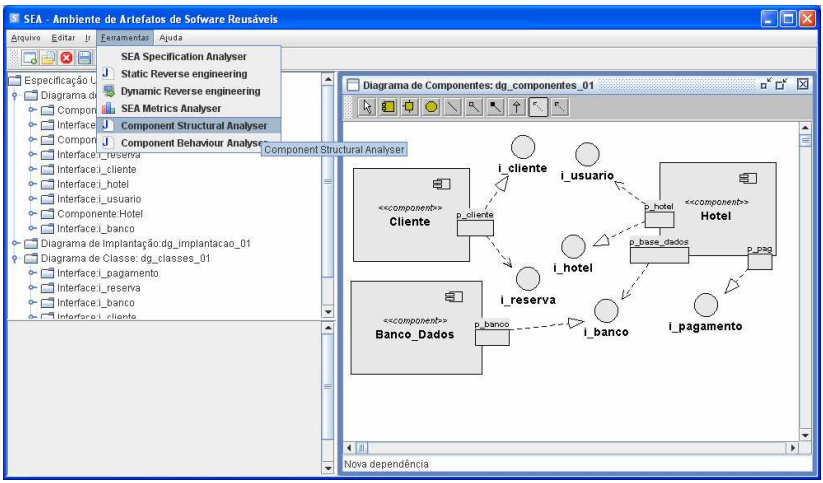


Figura 5.5: Ferramenta de Análise Estrutural do ambiente SEA.

A análise estrutural é composta de duas análises, conforme explicado no capítulo 4. A primeira analisa se a especificação está consistente com a abordagem proposta.

Na especificação inicial desse exemplo, percebe-se que o componente *Banco_Dados* não possui portos declarados e que o porto *p_base_dados* do componente *Hotel* não possui interfaces associadas, o que é considerado, pela abordagem proposta, inconsistência no modelo. Percebe-se também que a interface *i_pagamento*, declarada no diagrama de componentes, não foi definida no diagrama de classes. O que também é considerado inconsistência.

Os portos *p_base_dados* e *p_pag* não estão conectados a outros portos. Esse aspecto não é considerado inconsistência. É possível existir componentes com portos não conectados.

A figura 5.6 ilustra o relatório gerado pela análise de consistência da especificação estrutural do exemplo ilustrado nas figuras 5.2, 5.3 e 5.4.

Como a análise detectou inconsistência na especificação do sistema, é necessário corrigi-la e deixá-la sem incoerência para que a análise dos portos conectados seja realizada.

Consideremos então a nova especificação do sistema de reserva de hotel, corrigida e definida pelos diagramas ilustrados nas figuras 5.7, 5.8 e 5.9, que exibem os diagramas de componentes, classes e implantação. Um porto, *p_banco*, foi adicionado ao componente *Banco_Dados* e uma nova interface, *i_banco*, foi declarada no diagrama de componentes. As interfaces *i_banco* e *i_pagamento* foram definidas no diagrama de classes.

Esses diagramas obedecem às regras estabelecidas na abordagem - definidas e explicadas no capítulo 4 - e, portanto, ao acionar a ferramenta de análise estrutural para esse caso, nenhum erro será encontrado na análise de consistência da especificação estrutural. Assim, a análise de portos conectados é realizada para analisar a compatibilidade estrutural de componentes.

Algumas inconsistências foram propositalmente inseridas. Uma delas é o serviço *efetivarReserva()* requerido pelo componente *Hotel* e não fornecido pelo componente *Cliente*. O resultado da análise é ilustrado na figura 5.10.

Pelo relatório é possível verificar que foram identificados erros entre os serviços fornecidos e requeridos dos componentes *Cliente* e *Hotel*. No diagrama de classes do ambiente SEA, ilustrados nas figuras 5.3 e 5.8 não é possível visualizar os tipos de métodos de uma interface, assim como seus parâmetros e tipo de retorno. Esse tipo de informação é definida e visualizada em outras telas. As mesmas foram omitidas nesse capítulo. Tal omissão não compromete o entendimento da abordagem.

Para corrigir a incompatibilidade estrutural de componentes, o desenvolvedor deve analisar os serviços incompatíveis e alterar a especificação. Outra opção seria utilizar um adaptador, porém adaptação de componentes está fora do escopo desse trabalho.

```
**************************************************************************
 Modelo analisado: dg_componente_exe01
 Modelo do tipo: Diagrama de Componentes
**************************************************************************
--------------------------------------------------------------------------
Componente: Cliente
--------------------------------------------------------------------------
 Componente analisado sem erros.
--------------------------------------------------------------------------
Componente: Hotel
--------------------------------------------------------------------------
 Componente analisado sem erros.
--------------------------------------------------------------------------
Componente: Banco_dados
--------------------------------------------------------------------------
 Erro: Componente sem porto associado.
--------------------------------------------------------------------------
Porto: p_cliente
--------------------------------------------------------------------------
 Porto analisado sem erros.
--------------------------------------------------------------------------
Porto: p_hotel
--------------------------------------------------------------------------
 Porto analisado sem erros.
--------------------------------------------------------------------------
Porto: p_pag
--------------------------------------------------------------------------
 Porto analisado sem erros.
--------------------------------------------------------------------------
Porto: p_base_dados
--------------------------------------------------------------------------
 Erro: Porto não possui interface associada.
--------------------------------------------------------------------------
Interface: i_cliente
--------------------------------------------------------------------------
 Interface analisada sem erros.
--------------------------------------------------------------------------
Interface: i_reserva
--------------------------------------------------------------------------
 Interface analisada sem erros.
--------------------------------------------------------------------------
Interface: i_hotel
--------------------------------------------------------------------------
 Interface analisada sem erros.
--------------------------------------------------------------------------
Interface: i_usuario
--------------------------------------------------------------------------
 Interface analisada sem erros.
--------------------------------------------------------------------------
Interface: i_pagamento
--------------------------------------------------------------------------
Erro: Interface nao especificada em um Diagrama de Classes.
--------------------------------------------------------------------------
Modelo com erros.
--------------------------------------------------------------------------
**************************************************************************
 Modelo analisado: dg_implantação_Exe01
 Modelo do tipo: Diagrama de Componentes
**************************************************************************
Advertência: Porto [p_base_dados] do componente [Hotel] sem conexão.
             Porto [p_pag] do componente [Hotel] sem conexão.
--------------------------------------------------------------------------
Modelo com Advertência.
--------------------------------------------------------------------------
```

Figura 5.6: Relatório de análise de consistência da especificação estrutural com erros.

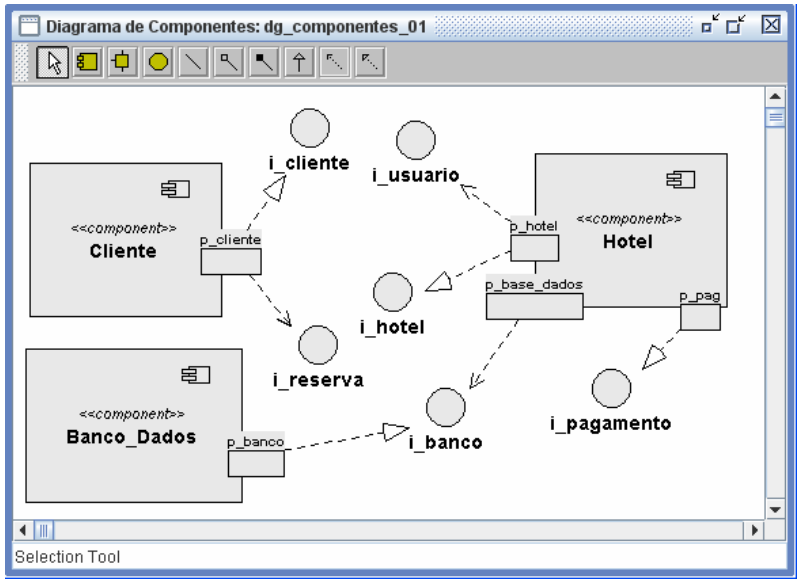


Figura 5.7: Diagrama de componentes sem erros.

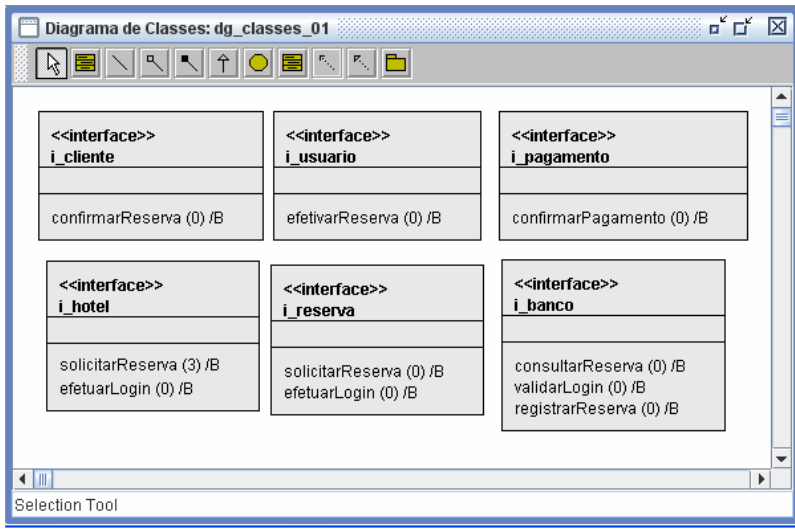


Figura 5.8: Diagrama de classes com todas as interfaces definidas.

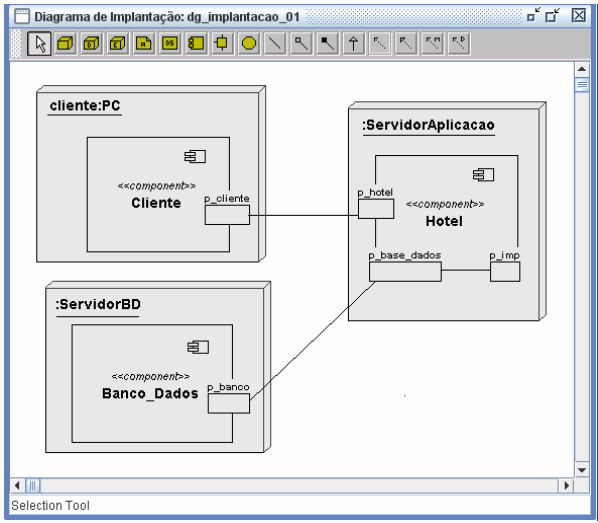


Figura 5.9: Diagrama de implantação com todos os componentes conectados.

5.3 CRIANDO A ESPECIFICAÇÃO COMPORTAMENTAL DE COMPONENTES NO AMBIENTE SEA

Conforme definido no capítulo 4, na abordagem utilizada nesse trabalho, cada componente deve ter sua máquina de estados associada, descrita pelo diagrama de máquina de estados. Ao criar um novo diagrama de máquina de estado, o ambiente pergunta se essa máquina de estados é associada a um componente. Se sim, uma lista dos componentes declarados em diagramas de componentes é exibida para seleção. O *framework* OCEAN é quem possibilita essa ligação entre conceitos e modelos.

Vamos considerar a situação definida inicialmente: no sistema de reserva de hotel, o cliente solicita uma reserva, o hotel registra essa reserva e a confirma para o cliente. O comportamento dos componentes é definido pelas máquinas de estados ilustradas na figura 5.11. O componente *Cliente*, por exemplo, solicita a reserva ao hotel, e aguarda a sua confirmação. O componente *Hotel*, ao receber a solicitação de uma reserva pelo cliente, solicita o registro da reserva ao componente *Banco_Dados*, e retorna a confirmação da reserva ao cliente.

```
********************************************************************************************
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Estrutural de Componentes
********************************************************************************************
--------------------------------------------------------------------------------------------
 Analisando Componente [Hotel] no porto [p_hotel]
 conectado ao Componente [Cliente] no porto [p_cliente]

- Servico [efetivarReserva(): void].
    NAO fornecido.
--------------------------------------------------------------------------------------------
 Analisando Componente [Cliente] no porto [p_cliente]
 conectado ao Componente [Hotel] no porto [p_hotel]

- Servico [solicitarReserva(): void].
    Fornecido como [solicitarReserva(Nome:String; Data: Date; NrDias: Integer): void].
    Quantidade de parametros diferente.
- Servico [efetuarLogin(User:String; Senha:String): Boolean].
    Fornecido como [efetuarLogin(User:String; Senha:String): String].
    Possui tipo de retorno diferente.
--------------------------------------------------------------------------------------------
 Analisando Componente [Hotel] no porto [p_base_dados]
 conectado ao Componente [Banco_Dados] no porto [p_banco]

- Servico [consultarReserva(): void].
    Servico [consultarReserva] encontrado OK.
- Servico [registrarReserva(): void].
    Servico [registrarReserva] encontrado OK.
- Servico [validarLogin(): void].
    Servico [validarLogin] encontrado OK.
--------------------------------------------------------------------------------------------
 Analisando Componente [Banco_Dados] no porto [p_banco]
 conectado ao Componente [Hotel] no porto [p_base_dados]

- Porto [p_banco] nao possui servicos requeridos.
--------------------------------------------------------------------------------------------
Analise Estrutural de Componentes com erros.
--------------------------------------------------------------------------------------------
Fim da Analise Estrutual.
--------------------------------------------------------------------------------------------
```

Figura 5.10: Relatório de análise estrutural (portos conectados).

5.4 REALIZANDO A ANÁLISE COMPORTAMENTAL DE COMPONENTES NO AMBIENTE SEA

A análise comportamental só deve ser realizada caso não haja erros de compatibilidade estrutural. Então, suponhamos que no sistema de reserva de hotel os problemas de compatibilidade estrutural tenham sido corrigidos na especificação. O serviço *efetivarReserva()* foi substituído pelo *confirmarReserva()*, fornecido pelo componente *Cliente*, e os parâmetros e tipos dos serviços *efetuarLogin()* e *solicitarReserva()* foram alterados. Dessa forma, tendo a especificação comportamental definida para cada componente, a análise comportamental pode ser executada.

Para realizar essa análise, é necessário acessar a opção do menu "*Ferramentas -> Component Behavioral Analyser*". A figura 5.12 ilustra a ferramenta de análise estrutural sendo acionada.

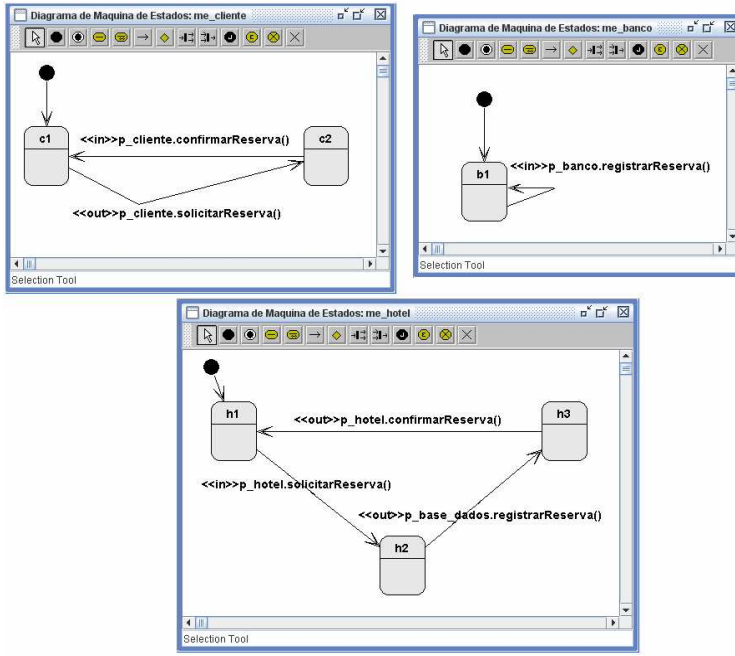


Figura 5.11: Máquinas de estados dos componentes Cliente, Hotel e Banco_Dados.

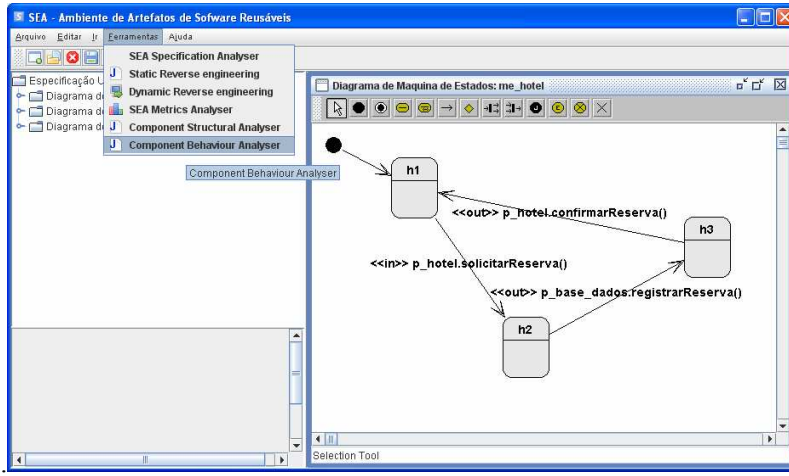


Figura 5.12: Ferramenta de Análise Comportamental do ambiente SEA.

5.4.1 Analisando a consistência da especificação comportamental.

A primeira análise realizada é a de consistência da especificação comportamental. Algumas das inconsistências verificadas são realizadas na edição do diagrama de máquina de estados, como por exemplo: sendo o diagrama vinculado a um componente, as transições só poderão ser rotuladas com portos e serviços associados àquele componente. A principal inconsistência realizada por essa análise é a verificação se foi definida a máquina de estados para cada componente pertencente à aplicação. Caso não seja encontrada a máquina de estados para algum componente, a análise comportamental não poderá ser realizada.

No exemplo do sistema de reserva de hotel, todas as máquinas de estados dos componentes envolvidos foram definidas e a análise não detectou erros. Devido à simplicidade do relatório gerado por essa análise, o mesmo foi omitido.

5.4.2 Gerando a máquina de estados da aplicação.

Inexistindo inconsistência na especificação, a máquina de estados da aplicação é gerada automaticamente pelo sistema e exibida ao usuário do ambiente SEA.

A figura 5.13 ilustra a máquina de estados da aplicação gerada a partir das máquinas de estados dos componentes *Cliente*, *Hotel* e *Banco_Dados,* ilustradas na figura 5.11.

5.4.3 Convertendo as máquinas de estados para as redes de Petri.

Gerada a máquina de estados da aplicação, todas as máquinas de estados são convertidas para redes de Petri, automaticamente. Esse modelo não é exibido ao usuário.

5.4.4 Interpretando as redes de Petri.

Geradas as redes de Petri, o analisador Pipe, integrado ao ambiente SEA, é acionado para verificar as propriedades de cada rede (binária, limitada, bloqueio e invariantes de transição). Conforme descrito no capítulo 4, seção 4.4.2.4, a análise de algumas propriedades ainda não está implementada (reiniciável, viva, transições quase vivas e transições mortas). Nesse caso, essas informações são inseridas

manualmente. As propriedades são interpretadas e o relatório de compatibilidade comportamental é gerado e disponibilizado ao usuário.

A figura 5.14 ilustra o relatório de compatibilidade comportamental desse exemplo.

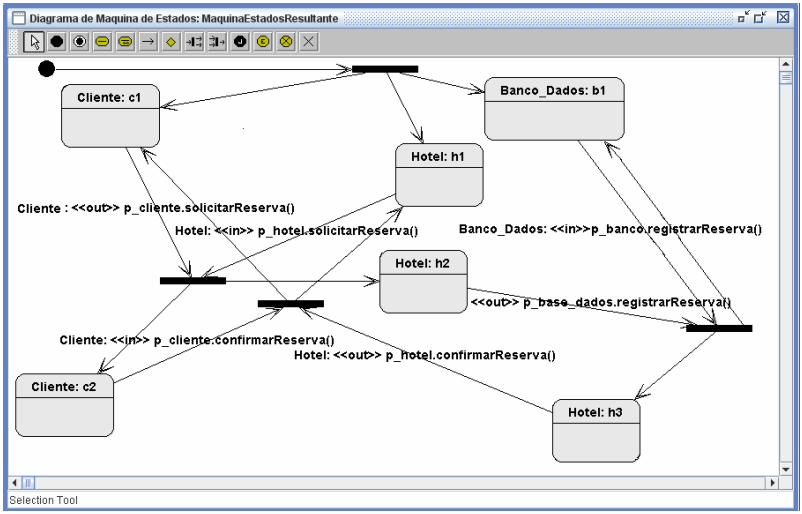


Figura 5.13: Máquina de estados da aplicação gerada a partir das máquinas de estados ilustradas na figura

```
-------------------------------------------------------------------------------------------------
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Comportamental
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Cliente]
-------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C1: solicitarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Hotel]
-------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C2: solicitarReserva(), registrarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Banco_Dados]
-------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C3: registrarReserva()
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Aplicação
-------------------------------------------------------------------------------------------------
- Aplicação com especificação ok.
- Ciclos de execucao de serviços
  C4: solicitarReserva(), registrarReserva(),  confirmarReserva()
-------------------------------------------------------------------------------------------------
 Quadro comparativo dos serviços dos Componentes
-------------------------------------------------------------------------------------------------
 Serviço/Comp          |       Cliente |     Hotel     |  Banco_Dados      | Aplicação
-------------------------------------------------------------------------------------------------
  solicitarReserva()   |disponivel(C1) |disponivel(C2)| -                  |disponivel(C4)
  registrarReserva()   |-              |disponivel(C2)| disponivel(C3)     |disponivel(C4)
  confirmarReserva()   |disponivel(C1) |disponivel(C2)| -                  |disponivel(C4)
-------------------------------------------------------------------------------------------------
Análise Comportamental OK
-------------------------------------------------------------------------------------------------
```

Figura 5.14: Relatório de análise de compatibilidade comportamental sem erros.

5.5 INSERINDO INCOMPATIBILIDADES NO SISTEMA

Nessa seção, vamos ilustrar algumas incompatibilidades que foram inseridas propositalmente para avaliar a proposta.

5.5.1 Identificando Advertências

Exemplo 01: Vamos considerar a inclusão do seguinte requisito: *login* do sistema - no sistema de reserva de hotel, para o cliente solicitar uma reserva é necessário que ele faça o *login* no sistema. O *login* é então validado e o cliente é habilitado a fazer a reserva. Portanto, os comportamentos dos componentes, descritos na figura 5.11, precisam ser alterados. Para esse novo exemplo, os comportamentos dos componentes são definidos pelas máquinas de estados ilustradas na figura 5.15. Observe que nada foi alterado em relação à estrutura dos componentes. Somente o aspecto comportamental dos componentes sofreu alteração.

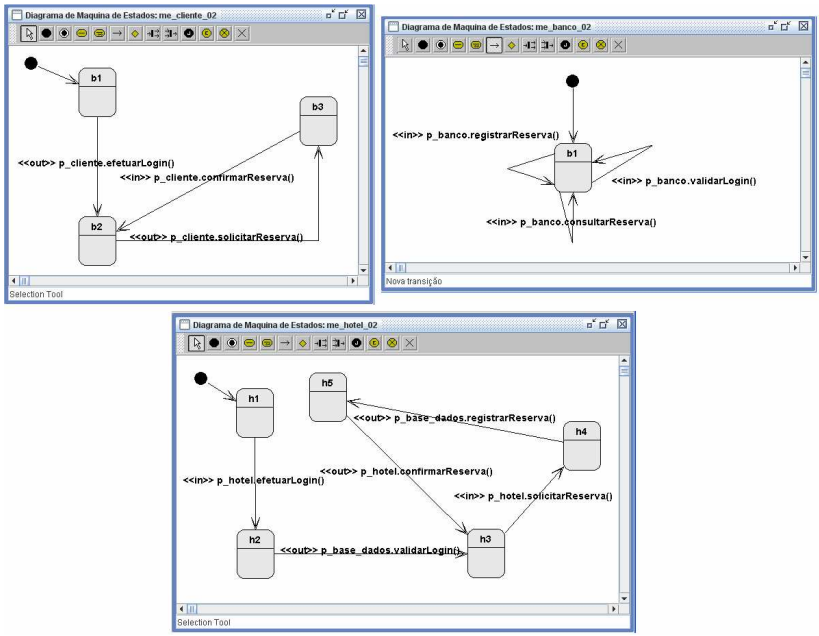


Figura 5.15: Máquina de estados dos componentes do exemplo 01.

Ao acionar a ferramenta de análise comportamental, a máquina de estados da aplicação é gerada e exibida ao usuário. A figura 5.16 ilustra a máquina de estados da aplicação desse exemplo e a figura 5.17 ilustra o relatório de análise comportamental gerado ao final da análise.

A análise identificou algumas advertências nos componentes e na aplicação. Por exemplo, o serviço *efetuarLogin()* é temporariamente disponível nos componentes *Cliente* e *Hotel* e também na aplicação. Os componentes *Cliente*, *Hotel* e a aplicação não retornam ao seu estado inicial.

Essas situações necessitam da avaliação do usuário para identificar se comprometem o funcionamento do sistema modelado ou não. Informações adicionais poderiam ser inseridas na especificação do sistema para permitir a identificação automática de erros nessas situações, como por exemplo: o componente deve retornar ao seu estado inicial (S/N?). Esse trabalho não tratou esse tipo de melhoria.

5.5.2 Identificando erro devido a portos não conectados

Exemplo 02: Agora, vamos considerar a inclusão do seguinte requisito: confirmação de pagamento - no sistema de reserva de hotel, para que o cliente receba a confirmação da reserva, é necessário que o hotel receba a confirmação do pagamento primeiro, ou seja, o registro da reserva só pode ser realizado com a confirmação do pagamento.

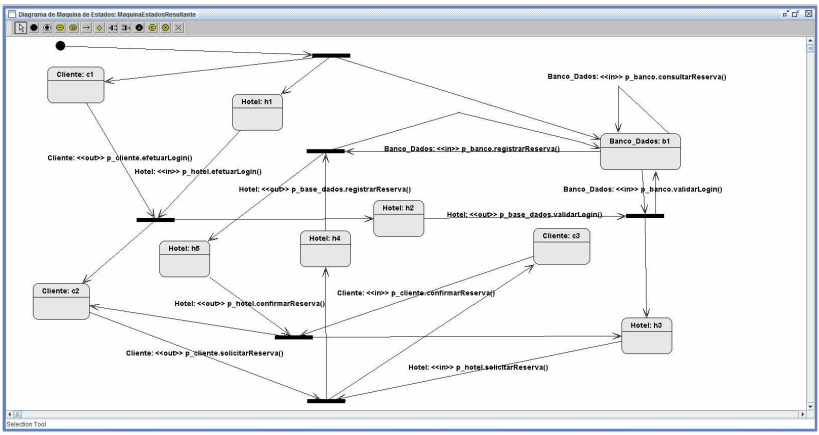


Figura 5.16: Máquina da aplicação do exemplo 01.

```
-------------------------------------------------------------------------------------------------
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Comportamental
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Cliente]
-------------------------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] requerido pelo porto [p_cliente].
- Ciclos de execução de serviços
  C1: solicitarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Hotel]
-------------------------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] fornecido pelo porto [p_hotel].
            [validarLogin()] requerido pelo porto [p_base_dados].
- Ciclos de execução de serviços
  C2: solicitarReserva(), registrarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Banco_Dados]
-------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C3: registrarReserva()
  C4: validarLogin()
  C5: consultarReserva()
-------------------------------------------------------------------------------------------------
 Comportamento analisado: Aplicação
-------------------------------------------------------------------------------------------------
- Advertência: Aplicação não reiniciável.
- Advertência: aplicação com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] requerido pelo componente [Cliente] no porto [p_cliente] e
                             fornecido pelo componente [Hotel] no porto [p_hotel].
            [validarLogin()] requerido pelo componente [Cliente] no porto [p_base_dados] e
                             fornecido pelo componente [Banco_Dados] no porto [p_banco].
- Advertência: aplicação com serviços indisponíveis.
  Serviços: [consultarReserva()] fornecido pelo componente [Banco_Dados] no porto [p_banco].
- Ciclos de execucao de serviços
  C6: solicitarReserva(), registrarReserva(),  confirmarReserva()
-------------------------------------------------------------------------------------------------
 Quadro comparativo dos serviços dos Componentes
-------------------------------------------------------------------------------------------------
 Serviço/Comp           |       Cliente |     Hotel     |  Banco_Dados  || Aplicação
-------------------------------------------------------------------------------------------------
  efetuarLogin()        |temp.disponivel|temp.disponivel|-              |temp.disponivel
  validarLogin()        |-              |temp.disponivel|disponivel(C4) |temp.disponivel
  solicitarReserva()    |disponivel(C1) |disponivel(C2) |-              |disponivel(C6)
  registrarReserva()    |-              |disponivel(C2) |disponivel(C3) |disponivel(C6)
  confirmarReserva()    |disponivel(C1) |disponivel(C2) |-              |disponivel(C6)
  consultarReserva()    |-              |-              |disponivel(C5) |indisponivel
-------------------------------------------------------------------------------------------------
Advertência: Ciclo [C4] disponivel no componente [Banco_Dados] e
                        temp.disponivel na Aplicacao.
             Ciclo [C5] disponivel no componente [Banco_Dados] e
                        indisponivel na Aplicacao.
-------------------------------------------------------------------------------------------------
Análise Comportamental com Advertências
-------------------------------------------------------------------------------------------------
```

Figura 5.17: Relatório de análise comportamental do exemplo 01.

O componente *Hotel* pode receber essa informação por meio do serviço *confirmarPagamento()* fornecido pelo porto *p_pag*. Para esse novo exemplo o aspecto comportamental dos componentes *Cliente* e *Banco_Dados* não sofreram alteração, somente o comportamento do componente *Hotel* foi modificado. A figura 5.18 ilustra o comportamento dos componentes para esse exemplo. Nenhuma alteração foi realizada na estrutura dos componentes.

A figura 5.19 ilustra a máquina de estados da aplicação, gerada após a ferramenta de análise comportamental ser acionada.

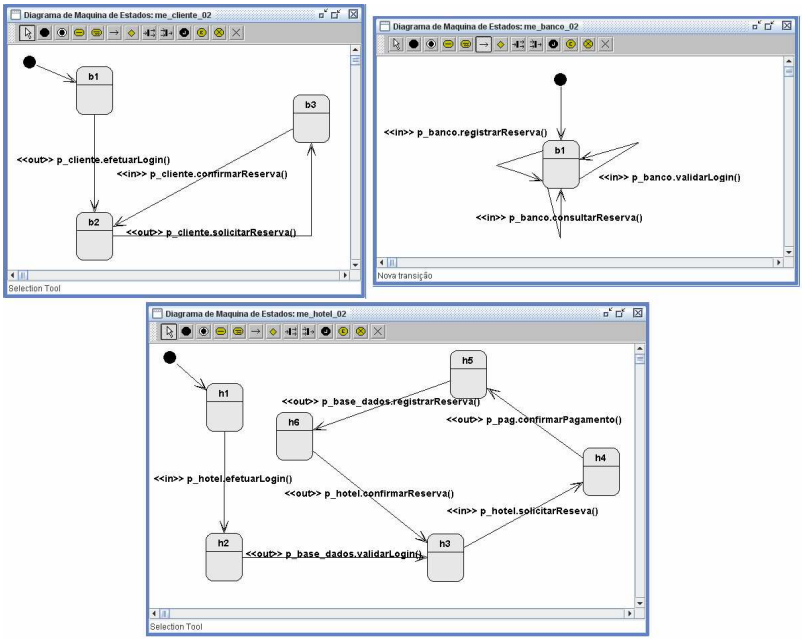


Figura 5.18: Máquina de estados dos componentes do exemplo 02.

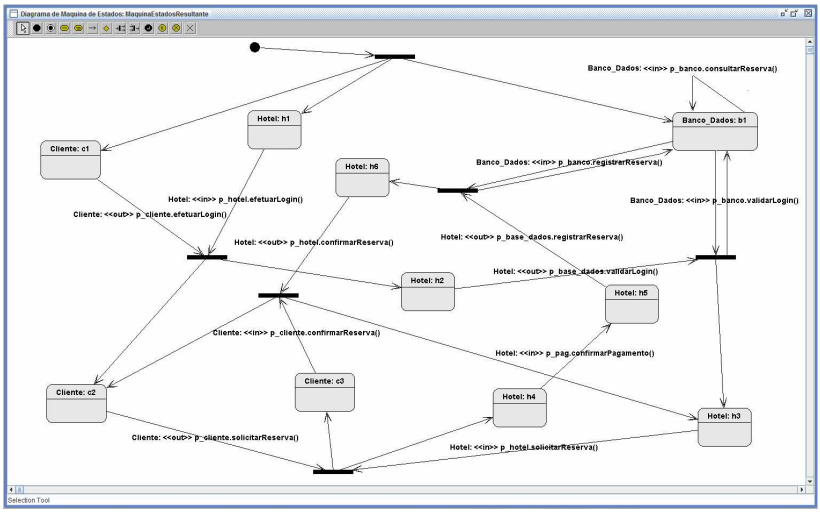


Figura 5.19: Máquina da aplicação do exemplo 02.

Ao final da análise, é gerado o relatório de análise comportamental, ilustrado na figura 5.20.

Nesse caso, a análise detectou advertência e erro. O erro identificado foi o bloqueio da aplicação devido a serviços essenciais à aplicação, fornecidos ou requeridos em portos não conectados. A análise detectou que o serviço *confirmarPagamento()*, fornecido pelo componente *Hotel* é essencial para o funcionamento da aplicação. Pelo diagrama de implantação, podemos comprovar que esse porto não está conectado a nenhum outro. Esse aspecto, apesar de ser identificado no relatório de análise estrutural, não configura erro estrutural, como dito anteriormente na seção 5.2.

Problemas relacionados a serviços essenciais para a aplicação, que são fornecidos ou requeridos por portos não conectados, são detectados somente através da análise comportamental.

```
-------------------------------------------------------------------------------------------------------
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Comportamental
-------------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Cliente]
-------------------------------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] requerido pelo porto [p_cliente].
- Ciclos de execução de serviços
  C1: solicitarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Hotel]
-------------------------------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] fornecido pelo porto [p_hotel].
            [validarLogin()] requerido pelo porto [p_base_dados].
- Ciclos de execução de serviços
  C2: solicitarReserva(), confirmarPagamento(), registrarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Banco_Dados]
-------------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C3: registrarReserva()
  C4: validarLogin()
  C5: consultarReserva()
-------------------------------------------------------------------------------------------------------
 Comportamento analisado: Aplicação
-------------------------------------------------------------------------------------------------------
- Advertência: Aplicação não reiniciável.
- Erro: Aplicação bloqueada devido a portos não conectados.
  - Porto [p_pag] do Componente [Hotel] não conectado com os seguintes
    serviços fornecidos essenciais à sua execução:[confirmarPagamento()]
- Ciclos de execução de serviços
  Não consta.
-------------------------------------------------------------------------------------------------------
 Quadro comparativo dos serviços dos Componentes
-------------------------------------------------------------------------------------------------------
 Serviço/Comp           |    Cliente    |     Hotel     |  Banco_Dados    | Aplicação
-------------------------------------------------------------------------------------------------------
  efetuarLogin()        |               |               |                 |
  validarLogin()        |               |               |                 |
  solicitarReserva()    |               |               |                 |
  registrarReserva()    |               |               |                 |
  confirmarReserva()    |               |               |                 |
  consultarReserva()    |               |               |                 |
  confirmarPagamento()  |               |               |                 |
-------------------------------------------------------------------------------------------------------
Análise Comportamental com Advertências e Erro.
-------------------------------------------------------------------------------------------------------
```

Figura 5.20: Relatório de análise comportamental do exemplo 02.

5.5.3 Identificando erro devido à ordem de execução dos serviços.

Exemplo 03: Agora, vamos considerar a seguinte situação: o componente *Cliente* foi especificado para solicitar e receber a confirmação da reserva, e não trata o requisito *login* do sistema. Porém, o componente *Hotel* foi especificado para aceitar a reserva do cliente somente depois do mesmo realizar o *login*.

Os novos comportamentos foram definidos conforme ilustra a figura 5.21. Após acionar a ferramenta de análise comportamental, o comportamento do sistema é exibido, conforme ilustra a figura 5.22, e o relatório de análise comportamental gerado, conforme ilustra a figura 5.23.

Nesse exemplo a análise também detectou advertência e erro. Porém o erro identificado foi o bloqueio da aplicação devido às restrições associadas à ordem de execução dos métodos estabelecidas por um componente não serem respeitadas pelo outro. No caso, o componente *Cliente* invoca o serviço *solicitarReserva()* do componente *Hotel*, que está preparado para receber a invocação do método *efetuarLogin()*.

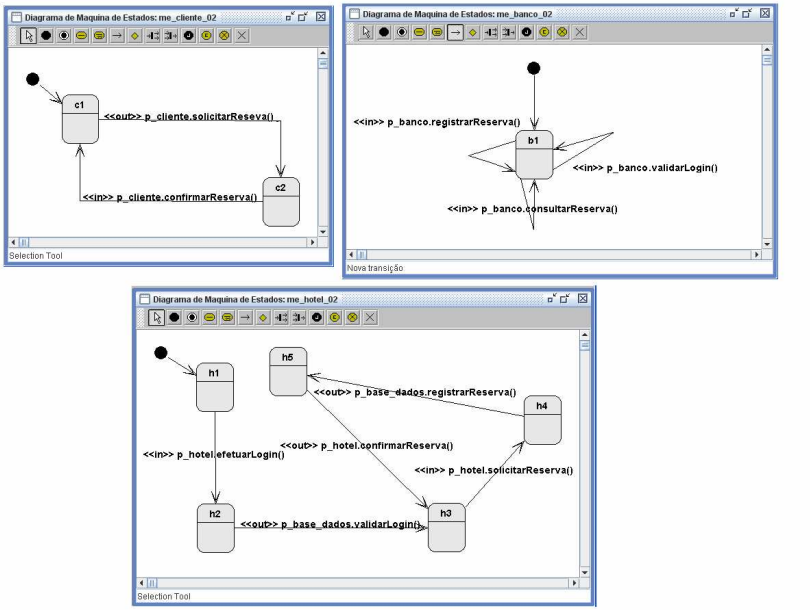


Figura 5.21: Máquina de estados dos componentes do exemplo 03.

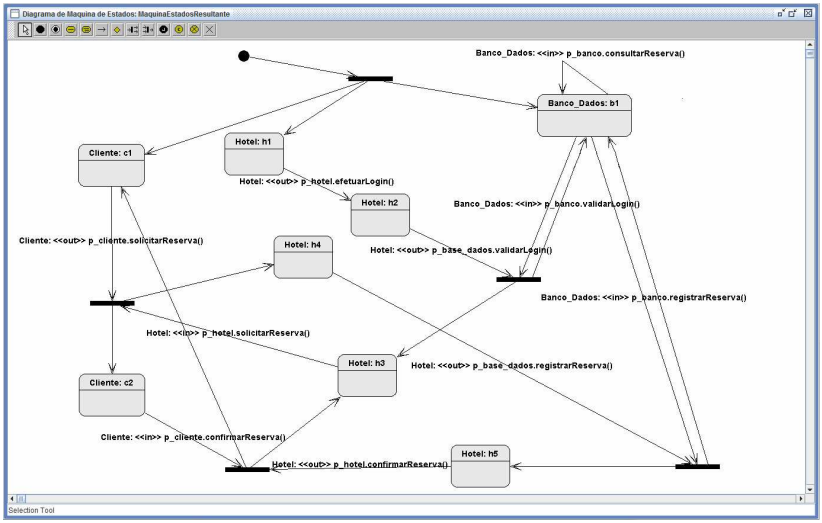


Figura 5.22: Máquina da aplicação do exemplo 03.

```
--------------------------------------------------------------------------------
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Comportamental
--------------------------------------------------------------------------------
 Comportamento analisado: Componente [Cliente]
--------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C1: solicitarReserva(), confirmarReserva()
--------------------------------------------------------------------------------
 Comportamento analisado: Componente [Hotel]
--------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] fornecido pelo porto [p_hotel].
            [validarLogin()] requerido pelo porto [p_base_dados].
- Ciclos de execução de serviços
  C2: solicitarReserva(), registrarReserva(), confirmarReserva()
--------------------------------------------------------------------------------
 Comportamento analisado: Componente [Banco_Dados]
--------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C3: registrarReserva()
  C4: validarLogin()
  C5: consultarReserva()
--------------------------------------------------------------------------------
 Comportamento analisado: Aplicação
--------------------------------------------------------------------------------
- Advertência: Aplicação não reiniciável.
- Erro: Aplicação bloqueada.
- Ciclos de execução de serviços
  Não consta.
--------------------------------------------------------------------------------
 Quadro comparativo dos serviços dos Componentes
--------------------------------------------------------------------------------
 Serviço/Comp          |    Cliente    |     Hotel     |  Banco_Dados    | Aplicação
--------------------------------------------------------------------------------
  efetuarLogin()       |               |               |                 |
  validarLogin()       |               |               |                 |
  solicitarReserva()   |               |               |                 |
  registrarReserva()   |               |               |                 |
  confirmarReserva()   |               |               |                 |
  consultarReserva()   |               |               |                 |
  confirmarPagamento() |               |               |                 |
--------------------------------------------------------------------------------
Análise Comportamental com Advertências e Erro.
--------------------------------------------------------------------------------
```

Figura 5.23: Relatório de análise comportamental do exemplo 03.

5.5.4 Identificando divergência de comportamento quando um componente é conectado a outros

Exemplo 04: Por último, vamos considerar a seguinte situação ainda sobre o requisito *login* do sistema: o cliente realiza o *login* no sistema antes de solicitar as reservas desejadas. O hotel identifica o cliente pelo *login*, independente da solicitação da reserva, mas também aceita a solicitação de reserva sem que haja necessidade de *login*.

Os aspectos comportamentais dos componentes são definidos conforme ilustra a figura 5.24. A máquina de estados da aplicação, gerada ao acionar a ferramenta de análise comportamental, é exibida na figura 5.25. E o relatório de análise comportamental desse exemplo é mostrado na figura 5.26.

Na análise comportamental não é identificado erro na aplicação, mas algumas advertências são encontradas.

Os serviços *efetuarLogin()* e *validarLogin()*, fornecido e requerido, respectivamente, pelo componente *Hotel*, são sempre disponível no comportamento do componente individual. Porém, na aplicação, esses serviços passam a ser temporariamente disponíveis.

O serviço *consultarReserva()*, fornecido pelo componente *Banco_Dados* está sempre disponível no comportamento desse componente. Na aplicação, esse serviço passa a ser indisponível, ou seja, ele nunca será executado.

Esse tipo de situação reflete mudança no comportamento do componente quando o mesmo é conectado a outros para formar uma aplicação. Nesses casos, a avaliação do usuário é essencial para definir se essas situações configuram erro ou não no sistema modelado.

Informações adicionais, referentes a esse tipo de situação, poderiam ser inseridas na especificação do sistema. Esse trabalho não tratou esse tipo de melhoria.

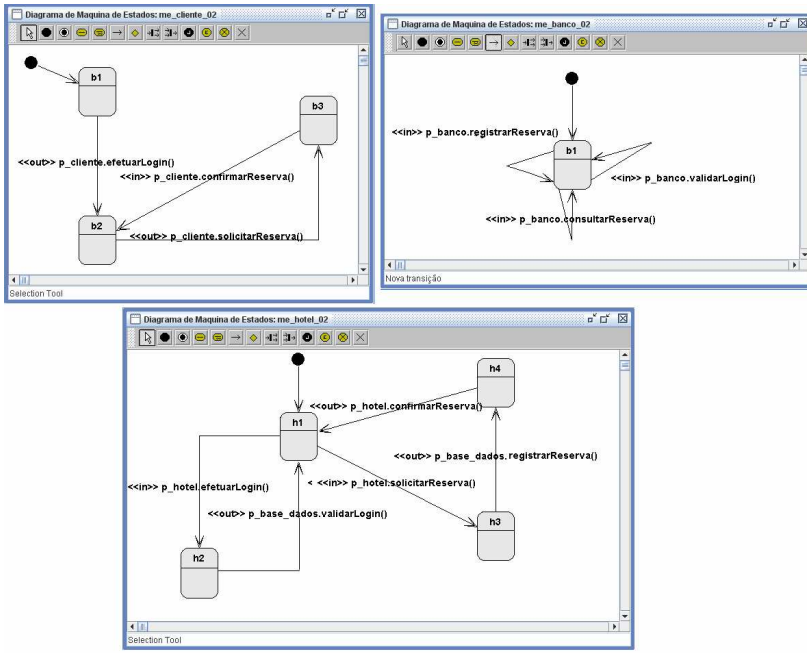


Figura 5.24: Máquina de estados dos componentes do exemplo 04.

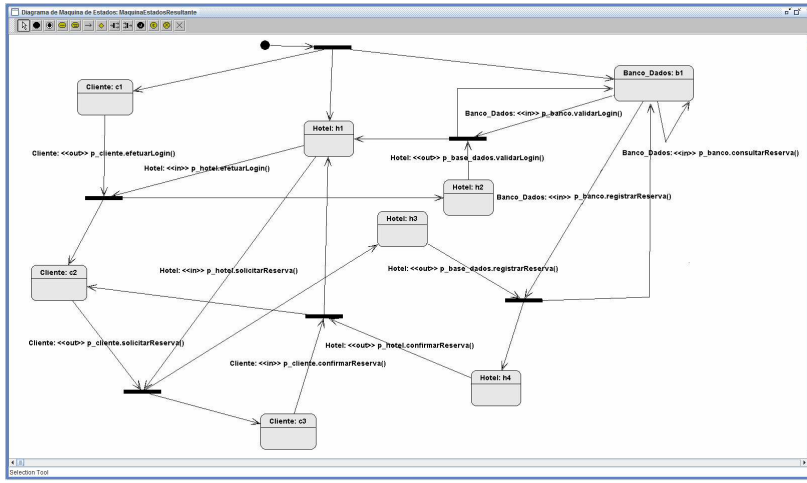


Figura 5.25: Máquina da aplicação do exemplo 04.

```
-------------------------------------------------------------------------------------------------------------
 Modelo Analisado: Componentes
 Analise do tipo: Compatibilidade Comportamental
-------------------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Cliente]
-------------------------------------------------------------------------------------------------------------
- Advertência: Componente não reiniciável.
- Advertência: Componente com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] requerido pelo porto [p_cliente].
- Ciclos de execução de serviços
  C1: solicitarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Hotel]
-------------------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C2: efetuarLogin(), validarLogin()
  C3: solicitarReserva(), registrarReserva(), confirmarReserva()
-------------------------------------------------------------------------------------------------------------
 Comportamento analisado: Componente [Banco_Dados]
-------------------------------------------------------------------------------------------------------------
- Componente com especificação ok.
- Ciclos de execução de serviços
  C4: registrarReserva()
  C5: validarLogin()
  C6: consultarReserva()
-------------------------------------------------------------------------------------------------------------
 Comportamento analisado: Aplicação
-------------------------------------------------------------------------------------------------------------
- Advertência: Aplicação não reiniciável.
- Advertência: aplicação com serviços temporariamente disponíveis.
  Serviços: [efetuarLogin()] requerido pelo componente [Cliente] no porto [p_cliente] e
                             fornecido pelo componente [Hotel] no porto [p_hotel].
            [validarLogin()] requerido pelo componente [Cliente] no porto [p_base_dados] e
                             fornecido pelo componente [Banco_Dados] no porto [p_banco].
- Advertência: aplicação com serviços indisponíveis.
  Serviços: [consultarReserva()] fornecido pelo componente [Banco_Dados] no porto [p_banco].
- Ciclos de execucao de serviços
  C7: solicitarReserva(), registrarReserva(),  confirmarReserva()
-------------------------------------------------------------------------------------------------------------
 Quadro comparativo dos serviços dos Componentes
-------------------------------------------------------------------------------------------------------------
 Serviço/Comp           |       Cliente |     Hotel     |  Banco_Dados  | Aplicação
-------------------------------------------------------------------------------------------------------------
  efetuarLogin()        |temp.disponivel|disponivel(C2) |-              |temp.disponivel
  validarLogin()        |-              |disponivel(C2) |disponivel(C5) |temp.disponivel
  solicitarReserva()    |disponivel(C1) |disponivel(C3) |-              |disponivel(C7)
  registrarReserva()    |-              |disponivel(C3) |disponivel(C4) |disponivel(C7)
  confirmarReserva()    |disponivel(C1) |disponivel(C3) |-              |disponivel(C7)
  consultarReserva()    |-              |-              |disponivel(C6) |indisponivel
-------------------------------------------------------------------------------------------------------------
Advertência: Ciclo [C2] disponivel no componente [Hotel] e
                        temp.disponivel na Aplicacao.
             Ciclo [C5] disponivel no componente [Banco_Dados] e
                        temp.disponivel na Aplicacao.
             Ciclo [C6] disponivel no componente [Banco_Dados] e
                        indisponivel na Aplicacao.
-------------------------------------------------------------------------------------------------------------
Análise Comportamental com Advertências
-------------------------------------------------------------------------------------------------------------
```

Figura 5.26: Relatório de análise comportamental do exemplo 04.

5.6 CRIANDO INSTÂNCIAS DE UM COMPONENTE

A abordagem adotada nesse trabalho, para a especificação de sistemas baseados em componentes, permite definir o componente uma única vez, em diagrama de componentes, assim como seu comportamento, em diagrama de máquina de estados. Porém, é possível utilizá-lo, em forma de instância, quantas vezes for necessário, no diagrama de implantação.

Consideremos como exemplo um jogo de tabuleiro qualquer[10] que deva ser jogado por 2 jogadores. Um jogador deve iniciar o jogo, enquanto o outro é convidado para participar da partida.

Iniciado o jogo, o primeiro jogador faz sua jogada e em seguida, o segundo jogador. Após cada jogada, é verificado se houve ganhador ou se a partida foi finalizada sem ganhador. Em qualquer um dos casos, os jogadores são avisados do fim da partida e estão aptos a iniciar uma nova. Caso contrário, podem realizar novas jogadas.

O jogo em questão é modelado com dois componentes: *Jogador* e *Jogo*, conforme ilustrado na figura 5.27. A organização dos componentes é definida no diagrama de implantação, ilustrado na figura 5.28. Para que o jogo funcione corretamente, duas instâncias do componente *Jogador* são necessárias. O componente *Jogo* possui dois portos onde deve ser conectada cada uma das instâncias de jogadores.

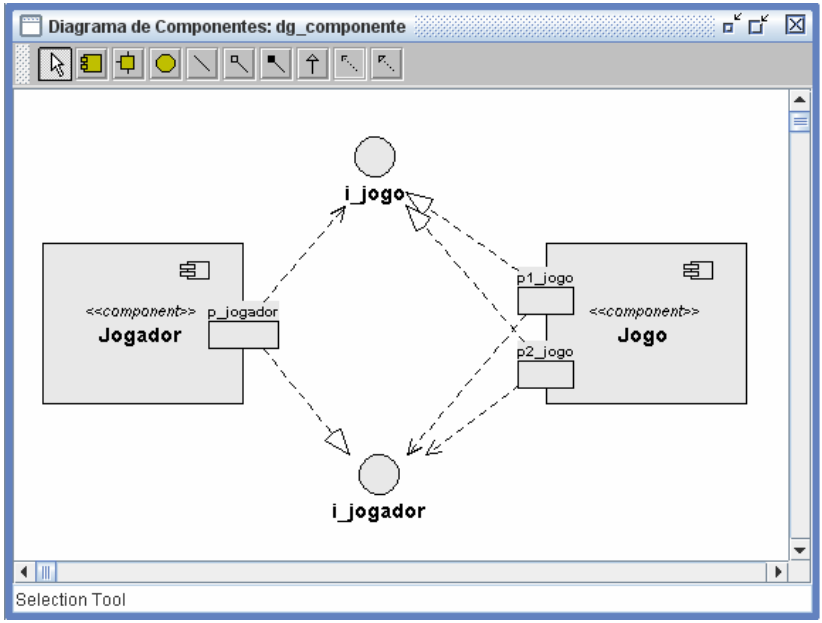


Figura 5.27: Diagrama de componentes do exemplo de jogo de tabuleiro.

[10] Aqui foi considerado um jogo de tabuleiro qualquer porque a lógica do jogo não interfere na ilustração do exemplo.

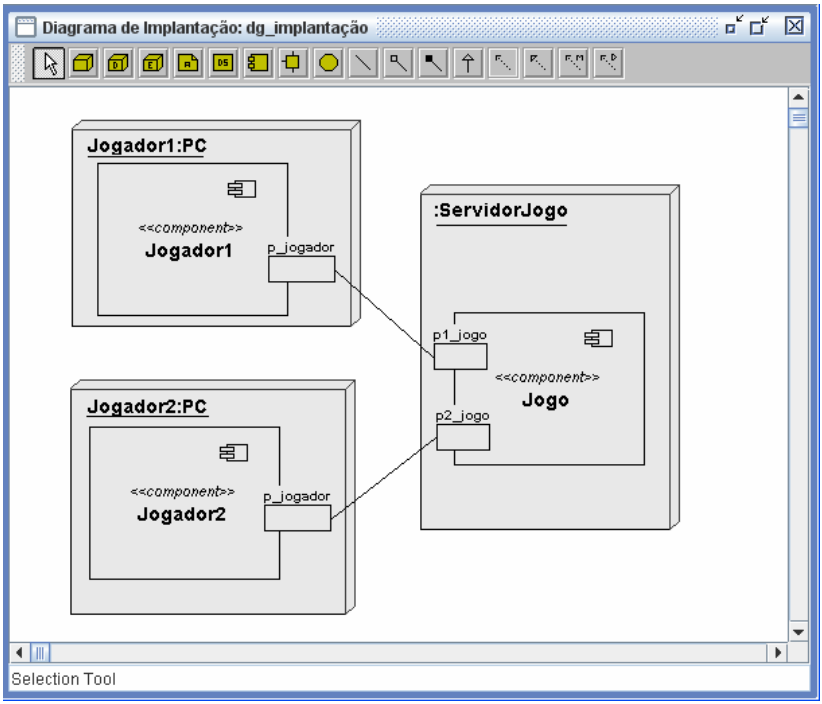


Figura 5.28: Diagrama de implantação do exemplo jogo de tabuleiro.

É necessário definir o comportamento dos dois componentes declarados no diagrama de componentes (*Jogador* e *Jogo*). Perceba que o comportamento do componente *Jogador* é definido uma única vez, embora haja duas instâncias desse componente no diagrama de implantação.

As figuras 5.29 e 5.30 ilustram o comportamento dos componentes *Jogador* e *Jogo* respectivamente. O componente *Jogador* pode iniciar o jogo ou receber um convite para participar de uma partida. Após ser habilitado, ele realiza sua jogada, e aguarda sua vez de jogar novamente, ou é notificado do fim da partida. O componente Jogo recebe a solicitação para iniciar a partida por meio do jogador conectado ao seu porto *p1_jogo*. Em seguida, convida o jogador conectado ao porto *p2_jogo*. O primeiro jogador é habilitado a realizar sua jogada, e depois o segundo jogador. A cada jogada, é verificado se a partida chegou ao final. Se sim, o jogador conectado ao porto *p1_jogo* é notificado do fim da partida e em seguida o jogador conectado ao porto

p2_jogo também é notificado. Ao final da partida, uma nova pode ser iniciada.

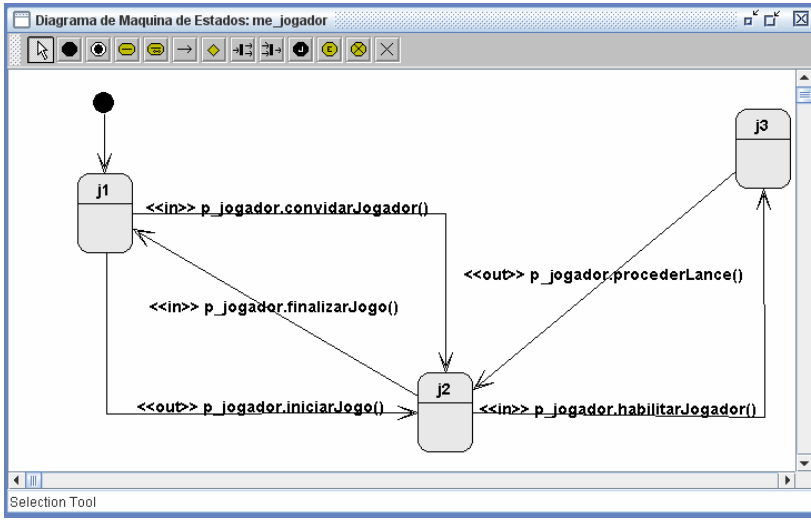


Figura 5.29: Máquina de estados do componente Jogador.

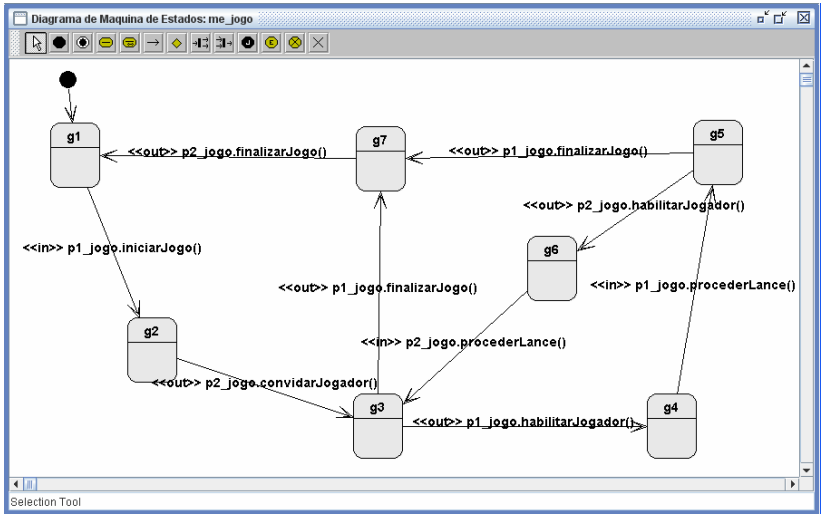


Figura 5.30: Máquina de estados do componente Jogo.

A análise estrutural foi executada para esse exemplo e não foi detectado erro (relatórios dessa análise para esse exemplo foram omitidos). Em seguida, foi realizada a análise comportamental. A figura 5.31 ilustra a máquina da aplicação gerada para esse exemplo e a figura 5.32 ilustra o relatório de análise comportamental.

Esse exemplo permite identificar que o comportamento de um componente pode ser diferente dependendo do porto em que ele é conectado.

O relatório de análise comportamental identificou que o serviço *convidarJogador(),* fornecido pelo componente *Jogador* fica indisponível na aplicação quando o mesmo é conectado pelo seu único porto ao porto *p1_jogo* do componente *Jogo* (instância Jogador1). O mesmo ocorre com o serviço *iniciarJogo()*, requerido pelo componente *Jogador*, quando conectado pelo seu único porto ao porto *p2_jogo* do componente *Jogo (instância Jogador2)*.

A análise se esse tipo de alteração no comportamento do componente, dependendo do porto a que ele é conectado, representa um problema para a aplicação deve ser realizada pelo usuário.

5.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo ilustrou estudos de caso para demonstrar a abordagem proposta e as análises de compatibilidade realizadas entre componentes. Os sistemas apresentados foram um sistema de reserva de hotel, contendo três componentes, e um jogo de tabuleiro, contendo dois componentes.

Primeiramente, foi criada a especificação estrutural de componentes, para então acionar a ferramenta de análise estrutural. Incompatibilidades estruturais foram inseridas na especificação para demonstrar as análises realizadas, que são listadas em um relatório ao final da análise.

Posteriormente, foi demonstrado como deve ser criada a especificação comportamental de cada componente. A partir de então, a ferramenta de análise comportamental pode ser acionada.

As seguintes situações comportamentais foram exemplificadas:

- especificação sem erro nem advertências;
- com advertências, incluindo alteração no comportamento do componente quando o mesmo é conectado para formar a aplicação;

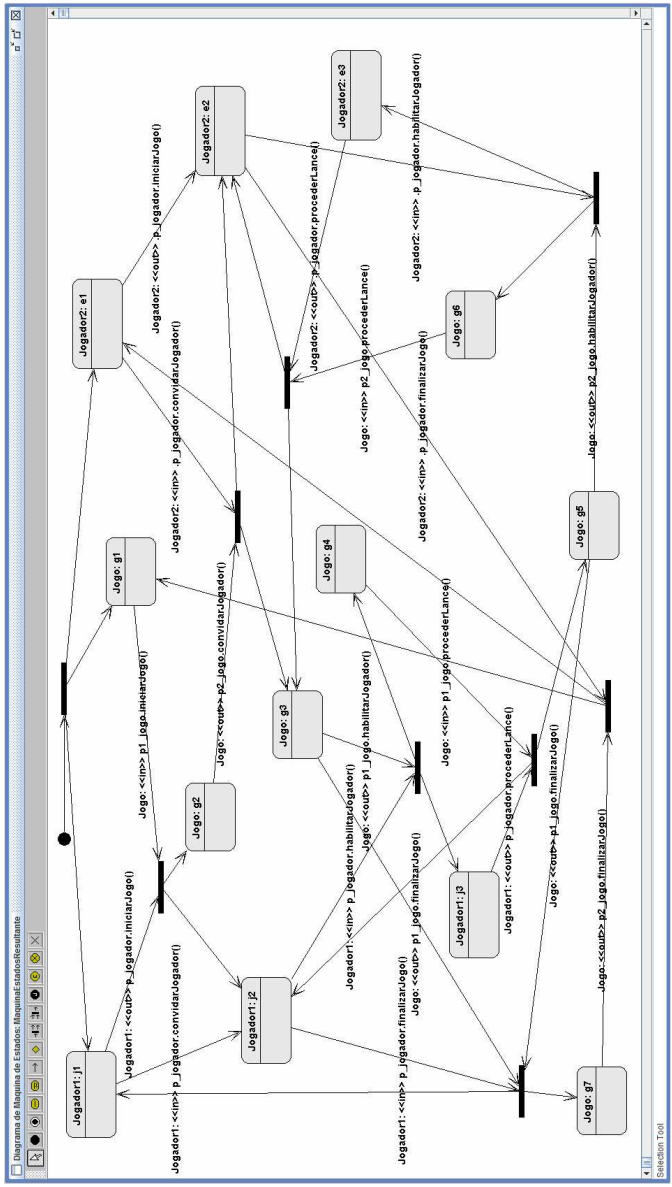


Figura 5. 31: Máquina de estados da aplicação do exemplo Jogo.

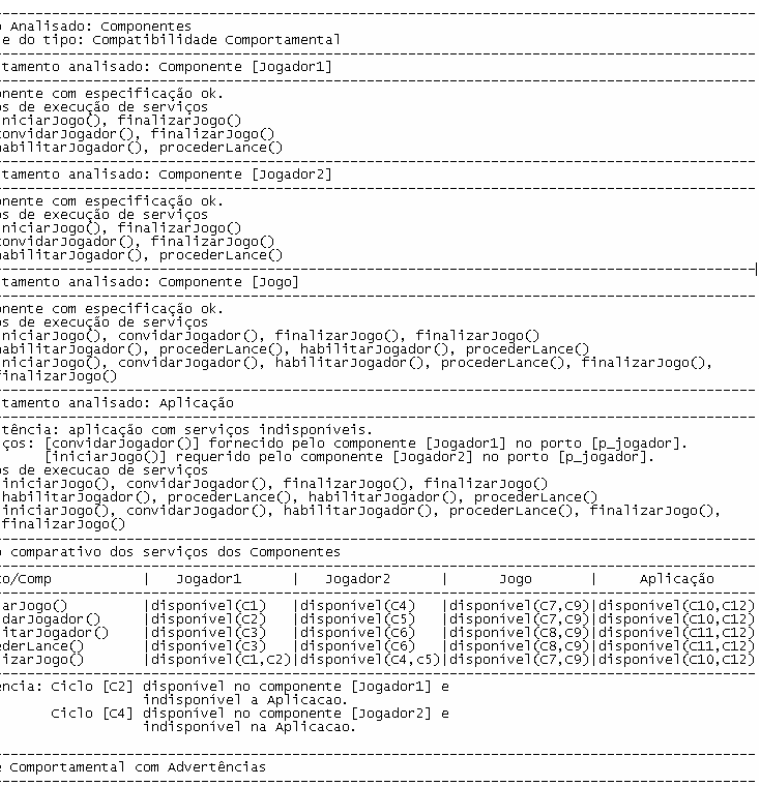


Figura 5.32: Relatório de análise comportamental do exemplo Jogo.

- com erro, ocasionado por serviços essenciais, fornecidos ou requeridos, em portos não conectados; e

- com erro devido ao não sincronismo na ordem de execução dos serviços.

Foram exemplificadas também situações em que mais de uma instância do componente pode ser criada. Nesse caso, o comportamento do componente pode ser diferente, dependendo do porto do outro componente a ele conectado.

Como dito no início do capítulo, o exemplo do sistema reserva de hotel é uma adaptação do exemplo apresentado em Chouali, Souquières e Heisel (2006), que também utiliza UML para a especificação dos componentes. Porém, o mesmo difere da abordagem desta dissertação quanto à forma como a especificação dos componentes

é feita. Além disso, o citado trabalho utiliza o método formal B para realizar as análises de compatibilidade.

Mesmo utilizando técnicas diferentes é possível realizar algumas comparações[11] entre os exemplos utilizados no trabalho citado e esta dissertação. A tabela 5.1 resume as principais diferenças encontradas. De uma forma geral, o exemplo desta dissertação tratou uma quantidade maior de requisitos e situações possíveis de serem especificadas no sistema abordado e identificou situações comportamentais não previstas no trabalho de Chouali, Souquières e Heisel (2006).

Desta forma, a proposta estabelecida neste trabalho para a análise comportamental de componentes mostrou-se eficaz para a identificação de problemas comportamentais relacionados à interligação dos componentes conectados em uma aplicação. Além de *deadlock*, e da identificação de sua causa, é possível encontrar situações suspeitas que podem significar erro na aplicação. Essas situações foram tratadas como advertências nesse trabalho e devem ser analisadas pelo usuário.

Tabela 5.1: Diferenças do exemplo sistema de reserva de hotel.

Aspecto comparado	**Chouali, Souquières e Heisel (2006)**	**Esta dissertação**
Quanto à especificação.	A especificação utilizada não trata o conceito porto. Desta forma, assume-se que os componentes especificados possuem apenas um porto, o que torna a abordagem mais limitada do que aquelas que tratam portos.	Trata o conceito porto. Um componente pode ter mais de um porto e a cada porto podem ser associadas várias interfaces.
	O exemplo possui dois componentes: *HotelMgr* e *ReservationSystem,* que correspondem aos componentes: *Cliente* e *Hotel* do exemplo desta dissertação.	O exemplo possui três componentes: *Cliente, Hotel e Banco_Dados*.

[11] Aqui, comparou-se essa dissertação com o trabalho de Chouali, Souquières e Heisel (2006) devido à semelhança do exemplo utilizado no estudo de caso. Esta dissertação também é comparada com os trabalhos correlatos apresentados no capítulo 3, tabela 3.2.

Quanto à análise de compatibilidade	A análise estrutural é realizada entre duas interfaces.	A análise estrutural é realizada entre par de portos conectados. Nesse caso a análise é mais complexa.
	A análise comportamental não é realizada[12].	A análise comportamental é realizada e considera o comportamento do conjunto de componentes. Além de deadlock, outros possíveis probelmas comportamentais são identificados.

[12] O trabalho de Mouakher, Lanoix e Souquières (2006) utiliza a mesma técnica do trabalho de Chouali, Souquières e Heisel (2006) e trata a análise comportamental. Porém, a única situação identificada na análise comportamental é deadlock. Além disso, o comportamento do conjunto de componentes não é tratado, como faz esta dissertação.

6 CONCLUSÕES, CONTRIBUIÇÕES E PERSPECTIVAS FUTURAS

Esse capítulo apresenta as conclusões, contribuições e os trabalhos futuros decorrentes desta dissertação.

6.1 CONCLUSÕES

A engenharia de software baseada em componentes tem estabelecido um novo segmento no desenvolvimento de aplicações. Porém, muitas questões relacionadas ao desenvolvimento de software baseado em componentes ainda estão em aberto, o que dificulta a construção de software utilizando esse paradigma.

Uma das questões discutidas em pesquisas é a de como descrever a interface de componentes. Várias abordagens são propostas com a finalidade de encontrar um padrão de descrição para as interfaces de componentes. Entretando, esse padrão ainda não foi estabelecido, o que dificulta o reuso dos componentes. Essa questão é fundamental para a abordagem de componentes, pois é a partir da descrição da interface que é possível analisar e decidir pelo uso ou não de um componente em uma aplicação baseada em componentes.

Silva (2009) propôs uma abordagem para especificação de componentes e de sistemas baseado em componentes, utilizando exclusivamente diagramas UML. Nessa proposta os componentes são definidos em diagramas de componentes, e sua interface inclui também o diagrama de classes. O comportamento dos componentes é descrito por meio do diagrama de máquina de estados, com extensões propostas pelo autor. Cada componente tem uma máquina de estados associada a ele. A organização dos componentes em uma aplicação é definida por meio do diagrama de implantação. Tendo a especificação comportamental de cada componente envolvido na aplicação e como eles são organizados, é possível gerar a especificação do comportamento da aplicação por meio da união das máquinas de estados de cada componente envolvido. Essa união resulta em uma única máquina de estados, chamada de máquina de estados da aplicação, que representa o comportamento da mesma.

Outra questão fundamental relacionada à abordagem diz respeito à compatibilidade de componentes. Para que uma aplicação baseada em componente funcione corretamente, os componentes conectados devem

ser compatíveis nos níveis estrutural, comportamental e funcional. Porém, mesmo conhecendo as características de um componente individualmente não há garantias do resultado da sua combinação com outros. Análises de compatibilidade são necessárias para prever possíveis problemas relacionados à conexão de componentes. Essas análises são realizadas a partir da descrição da interface de componentes.

Este trabalho teve como foco a especificação de novos componentes de software e a análise de compatibilidade - estrutural e comportamental - entre eles em uma aplicação baseada em componentes, no paradigma orientado a objetos. A análise de compatibilidade do nível funcional está fora do escopo deste trabalho.

Para a especificação de novos componentes, a proposta de Silva (2009) pode ser avaliada por meio do ambiente SEA, construído a partir do *framework* OCEAN. Melhorias[13] relacionadas ao suporte gráfico dos diagramas de componentes, classes, máquina de estados e implantação foram realizadas nesse trabalho, assim como uma nova ferramenta para geração da especificação comportamental da aplicação baseada em componentes. Algumas adaptações, descritas na seção 4.3.2, foram propostas a fim de evoluir a abordagem sugerida.

Por meio de estudo bibliográfico foram identificados problemas estruturais e comportamentais relacionados à conexão de componentes. Em relação ao nível estrutural, percebeu-se que as análises são mais consolidadas se comparadas com o nível comportamental. Em relação a esse último nível, mais da metade dos trabalhos pesquisados trata apenas bloqueio do sistema (*deadlock)* e não informa a sua causa, o que dificulta a correção do mesmo.

O principal objetivo desse trabalho foi identificar problemas relacionados à compatibilidade comportamental de componentes interligados em uma aplicação baseada em componentes, utilizando a abordagem sugerida por Silva (2009).

Buscaram-se na literatura técnicas utilizadas para possibilitar a análise de compatibilidade de componentes. Com base nas técnicas pesquisadas, foi definida uma estratégia a fim de possibilitar as análises de compatibilidade comportamental de componentes a partir da especificação feita exclusivamente em UML.

---

[13] As melhorias realizadas no suporte gráfico dos diagramas dizem respeito à usabilidade e funcionalidade dos mesmos. Também foram necessárias alterações devido às extensões propostas por Silva (2009), descritas na seção 4.3.1.

Foi identificado que a análise estrutural poderia ser feita por meio da leitura dos diagramas UML utilizados. Para essa análise, foi implementada uma ferramenta chamada de ferramenta de análise estrutural (FAE) no ambiente SEA. Após a especificação estrutural da interface de componentes, a FAE pode ser executada automaticamente para produzir o relatório de análise estrutural. Esse relatório ilustra os problemas estruturais encontrados na especificação.

Para a análise comportamental foi definida uma estratégia que consiste na conversão do diagrama de máquina de estados – que representa o aspecto comportamental – para redes de Petri.

Métodos já existentes para converter especificações UML em redes de Petri não foram suficientes para serem utilizados nessa abordagem, devido à semântica específica da máquina de estados da aplicação, que representa o comportamento do sistema. Por esse motivo, um novo método foi concebido para realizar essa transformação.

Assim, as máquinas de estados dos componentes individuais e da aplicação são convertidas em redes de Petri de forma automática, por meio de uma ferramenta implementada no ambiente SEA, sem qualquer intervenção do usuário da ferramenta, que manipula exclusivamente diagramas UML.

A partir dessa conversão, a análise comportamental pode ser realizada baseada nas propriedades das redes de Petri. Para isso, buscou-se interpretar cada propriedade no contexto de componentes.

Essa interpretação permitiu a identificação de possíveis problemas comportamentais que podem ocorrer na conexão de componentes. Algumas situações, como redes com "*deadlock*" e "não binárias" são consideradas erros comportamentais. Outras situações, como redes "não reiniciáveis", "com transições quase vivas" ou "com transições mortas" caracterizam advertência, e devem ser analisadas pelo usuário. Ciclos de execução de serviços, dos componentes individuais e da aplicação, são identificados por meio dos invariantes de transição. Dessa forma, é possível comparar o comportamento do componente individualmente com o comportamento dele dentro da aplicação. Essa comparação é importante para identificar possíveis mudanças no comportamento do componente quando ele é conectado a outros. A identificação se isso é um problema para o sistema deve ser feita pelo usuário.

Para a análise comportamental, foi implementada uma ferramenta chamada de ferramenta de análise comportamental (FAC) no ambiente SEA. Depois de construída a especificação comportamental e identificado que não há erros na análise estrutural, a FAC pode ser

executada automaticamente para produzir o relatório de análise comportamental. Esse relatório ilustra os problemas comportamentais encontrados na especificação, como erros ou advertências. As advertências devem ser analisadas pelo usuário para definir se a situação é de fato um problema ou não para a aplicação.

A construção das ferramentas e a avaliação da abordagem foram viáveis devido ao *framework* OCEAN e o ambiente SEA. Porém, ainda há necessidade de melhorias relacionadas à parte gráfica da ferramenta. O ambiente SEA ainda é um protótipo e necessita de ajustes para se tornar uma ferramenta mais robusta. Testes de *stress* não foram realizados no ambiente.

6.2 CONTRIBUIÇÕES

Diante dos vários problemas que podem ocorrer na conexão de componentes em um sistema baseado em componentes, principalmente a não garantia quanto ao perfeito funcionamento da aplicação em que um componente é combinado com outros, esse trabalho buscou situações que pudessem representar risco à aplicação construída sob esse paradigma na tentativa de aumentar a previsibilidade da composição de componentes.

A partir da solução proposta para a análise de compatibilidade de componentes interligados, que é feita por meio da análise das propriedades das redes de Petri, considerando o contexto de componentes, evidencia-se a principal contribuição desta dissertação, que é a possibilidade de identificar, além de *deadlock*, sua causa e outros problemas comportamentais. Situações suspeitas, tratadas como advertências, são identificadas e apresentadas ao usuário para tomadas de decisão.

Quando identificado bloqueio no sistema, é verificado se o bloqueio decorre de serviços essenciais, fornecidos ou requeridos por portos não conectados. Essa informação adicional pode facilitar a resolução do problema pelo usuário. Além disso, o comportamento individual de cada componente é comparado com o comportamento dele quando interligado a outros para formar a aplicação. Serviços que estão sempre disponíveis no comportamento do componente podem se tornar temporariamente disponíveis ou até indisponíveis na aplicação. Alterações no comportamento dos componentes podem representar riscos à aplicação e devem ser analisadas pelo usuário.

Podem ser consideradas como contribuições secundárias as ferramentas de análise estrutural e comportamental, FAE e FAC, desenvolvidas no ambiente SEA para a automatização e avaliação da proposta. Cada uma dessas ferramentas inclui um conjunto de outras ferramentas necessárias para a emissão dos relatórios com os resultados das análises. Uma das ferramentas incluídas é a ferramenta para geração da máquina de estados da aplicação, que representa o comportamento do sistema.

Portanto, o objetivo principal deste trabalho foi atingido: a partir da estratégia definida para possibilitar a identificação automática de problemas relacionados à compatibilidade de componentes especificados em UML, é possível ter uma maior previsibilidade da combinação dos componentes interligados.

Em relação à hipótese da pesquisa verificou-se que a conversão do diagrama de máquina de estados da UML em redes de Petri permitiu a identificação de uma gama maior de problemas comportamentais, devido às propriedades desse modelo. Em relação à análise comportamental é possível a identificação de outros possíveis problemas, além de *deadlock*, identificados como advertências, e o motivo pelo qual ocorre o bloqueio no sistema.

Devido a estas contribuições, este trabalho foi publicado em três eventos:

1. *LA-WASP, 2011*[14]. V Latin American Workshop on Aspect-Oriented Software Development Advanced Modularization Technique. Qualis B5. (TEIXEIRA e SILVA, 2011a).
2. *ICSEA, 2011*. The Sixth International Conference on Software Engineering Advances. Qualis B4. (TEIXEIRA e SILVA, 2011c).
3. *SCCC, 2011*. XXX International Conference of the Chilean Computer Science Society. Qualis B2. (TEIXEIRA e SILVA, 2011b).

6.3 LIMITAÇÕES E PERSPECTIVAS FUTURAS

Este trabalho trata a abordagem de componentes no nível de especificação e não implementa componentes em uma linguagem específica. Os aspectos estrutural e comportamental foram tratados na

[14] No evento LA-WASP 2001 o artigo foi publicado na seção "*Poster Session*".

análise de compatibilidade de componentes. O aspecto funcional ficou fora do escopo do trabalho. A automatização das análises foi possível devido ao *framework* OCEAN e ao ambiente SEA. Para a análise da compatibilidade comportamental, o analisador da ferramenta PIPE foi integrado ao ambiente SEA. Porém, esse analisador considera apenas as propriedades binária, limitada, bloqueio e invariantes de transição. Para a análise das propriedades reiniciável, viva, transições quase vivas e transições mortas é necessário estender a implementação do analisador. Essas propriedades podem ser verificadas pela análise por enumeração das marcações. Alguns problemas identificados na análise comportamental são tratados como advertência porque necessitam da verificação do usuário. Para uma análise mais robusta, foram identificadas algumas melhorias na especificação dos componentes. Este trabalho sugere essas melhorias mas não as trata.

Essa pesquisa levanta a possibilidade de outras pesquisas possíveis para que a solução proposta para a análise de compatibilidade de componentes dentro do ambiente SEA seja evoluída.

Considerando toda a abordagem detalhada no capítulo 4 e as limitações deste trabalho, algumas perspectivas de trabalho são listadas a seguir:

- *Extensão da ferramenta Pipe:* a análise das propriedades: reiniciável, viva, transições quase vivas e transições mortas não é tratada pela ferramenta Pipe, que foi integrada ao ambiente SEA. Extensões são necessárias para que a análise comportamental seja totalmente automatizada no ambiente SEA.
- *Implementação dos componentes*: a partir da especificação feita em UML, a geração de código em alguma linguagem ou modelo de componente específico pode ser realizada de forma automática a partir do ambiente SEA.
- *Busca e seleção de componentes já existentes*: bibliotecas de componentes também podem ser criadas para explorar técnicas do desenvolvimento com componentes ou desenvolvimento com reuso, onde componentes já existentes e disponíveis são selecionados para compor uma nova aplicação.
- *Tratamento de adaptadores*: diante dos problemas de incompatibilidade de componentes interligados, identifi-

cados nas análises, tem-se a possibilidade de adotar abordagens de compatibilização para o correto funcionamento do sistema baseado em componentes. Já que componentes são artefatos caixa preta, a compatibilização deve considerar a impossibilidade de alterá-los internamente. Técnicas como a de empacotamento e colagem, podem ser adotadas para esse propósito.

- *Melhorias na especificação de componentes:* informações adicionais sobre restrições do sistema modelado baseado em componentes podem ser inseridas na especificação para que novas análises possam ser realizadas, juntamente com as análises já existentes e demonstradas nessa dissertação.
- *Aplicação prática da abordagem:* tendo o ambiente SEA com toda a abordagem implementada, a mesma pode ser utilizada, testada e avaliada em casos práticos e reais de forma exaustiva dentro de laboratórios de pesquisas ou em empresas parceiras. Isso possibilita possíveis melhorias na proposta.
- *Análise de compatibilidade funcional:* analisar se a solução proposta nessa dissertação pode melhorar a viabilização de uma possível análise de compatibilidade funcional entre componentes de uma aplicação dentro do ambiente SEA.

Diante do que foi exposto, conclui-se, portanto, que os objetivos do trabalho foram alcançados. A identificação de outros possíveis problemas comportamentais, como: serviços temporariamente disponíveis, serviços indisponíveis e a identificação da mudança do comportamento do componente quando o mesmo é conectado a outros fornece uma maior previsibilidade do resultado da conexão dos componentes. Espera-se que essa solução contribua para a melhoria da qualidade dos softwares baseado em componentes e minimize falhas em aplicações construídas sob esse paradigma.

REFERÊNCIAS

AMORIM, J. de. **Integração dos frameworks JHotDraw e OCEAN para a produção de objetos visuais a partir do framework OCEAN**. Dissertação de Mestrado, Florianópolis, UFSC. 2006.

ArgoUML. Disponível em: <http://argouml.tigris.org/>. Acesso em: 10 abril 2011.

Atelier-B. Disponível em: < http://www.atelierb.eu/index-en.php>. Acesso em: 10 abril 2011.

B Method. Disponível em <<http://www.bmethod.com/>>. Acesso em: 5 setembro 2011.

BARESI, L., PEZZÈ, M. **On formalizing UML with high-level petri nets**. In *Concurrent Object-Oriented Programming and Petri Nets: Advances in Petri Nets* Springer-Verlag New York, Secaucus, NJ, 276-304. 2001.

BRERETON, P. **Evolution of Component Based Systems**. In: *Proceedings of International Workshop on Component-Based Software Engineering.* 1999.Disponível em: <http://faculty.ksu.edu.sa/ghazy/CBD_MSc/Ref-25.pdf> Acesso em: 2 outubro 2011.

BROWN, A. W. **Component-Based Software Engineering**. [S.l.]: IEEE Computer Society Press, 1996. p. 7-15.

BRUEL, J. M.; OBER, I. **Components Modeling in UML 2**. Studia Univ. Babes-Bolyai, Informatica, Volume LI, Number 1, 2006. Disponível em: <http://www.cs.ubbcluj.ro/~studia-i/2006-1/09-BruelOber.pdf>. Acesso em: 10 setembro 2011.

CARDOSO, J.; VALETTE, R. **Redes de Petri.** Florianópolis: Ed. UFSC, 1997.

CHOUALI, S.; SOUQUIÈRES, J. **Verifying the compatibility of components interfaces using the B formal method.** In: *International Conference on Software Engineering Research and Pratice*. 2005.

CHOUALI, S.; SOUQUIÈRES, J.; HEISEL, M. **Proving Component Interoperability with B Refinement**. In: *International Workshop on Formal Aspect on Component Software*, H. R. Arabnia and H.Reza, Eds. CSREA Press, 2005, to appear in ENCTS 2006.

COELHO, A. **Reengenharia do Framework OCEAN**. Dissertação de Mestrado, Florianópolis, UFSC. 2007.

COUNCILL, W. T.; HEINEMAN, G. T. **Component-Based Software Engineering and its Elements**. Boston: Addison-Wesley, Boston, 2001.

CRAIG, D.C; ZUBEREK, W.M. **Petri Nets in Modeling Component Behavior and Verifying Component Compatibility.** *Int. Workshop on Petri Nets and Software Engineering, in conjunction with the 28-th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency*, Siedlce, Poland, June 25-26, 2007.

CRAIG, D. C.; ZUBEREK, W. M. **Modelling and Verification of Compatibility of Component Composition**. In: *Third Workshop on Modelling of Objects, Components, and Agents. Aarhus*, Denmark, 11-13 October 2004.

CUNHA, R. C. **Suporte à Análise de Compatibilidade Comportamental e Estrutural entre Componentes no Ambiente SEA.** Dissertação de Mestrado, Florianópolis, UFSC, 2005.

DIAS, M.S.; VIEIRA, M.E.R. **Software Architecture Analysis based on Statechart Semantics**. In: *Proceedings of the 10th Internacional Workshop on Software Specification and Design.* [S.1]: IEEE Computer Society, 2000.p.133.

FALKOWSKI, KERSTIN. **A component concept for scientific experiments – focused on versatile visual component assembling**. In: *Proceedings of the Fifteenth International Workshop on Component-Oriented Programming* (WCOP) 2010. Pages 32-38.

GROENDA, HENNING. **Certification of Software Component Performance Specifications**. In: *Proceedings of the Fourteenth International Workshop on Component-Oriented Programming (WCOP).* 2009. Pages 13-21.

HAREL, D. et al. **Statecharts: a visual formalism for complex systems.** Science od Computer Programming, [S.1.], n.8, 1987.

JENSEN, K. **Coloured Petri Nets: Basic Concepts, Analisys Methods and Practical Use**. Vol 1. Springer, 1997. 2.ed. 234p.

JOHNSON, R. **Do Components Exist**. Disponível em: <http://www.c2.com/cgi/wiki?DoComponentsExist>. Acesso em: 5 outubro 2011.

JUNG, MIN YANG. **A layered approach for identifying systematic faults of component-based software systems**. In: *Proceedings of the 16th international workshop on Component-oriented programming (WCOP)*. 2011. New York, NY, USA. Pages 17-24.

KING, P.; POOLEY, R. **Using UML to derive stochastic Petri net models.** In *Proceedings of the 15th UK Performance Engineering Workshop*, pages 45-56, Bristol, UK, 1999.

KONG, J. et al. **Specifying behavioral semantics of UML diagrams through graph transformations**. In: *Journal of Systems and Software*. Volume 82 Issue 2, pages 292-306, New York, NY, USA: Elsevier Science Inc. February, 2009.

KOFRON, JAN: .**Checking software component behavior using behavior protocols and Spin** In*: Proceedings of the 2007 ACM symposium on Applied computing*. 2007.

KOFRON, J; MACH, M; PLASIL, FRANTISEK**. Behavior protocol verification: Fighting state explosion**. *International Journal of Computer and Information Science*, 6(1), 2005.

LEVESON, N.G. **Safeware: system safety and computers.** ACM New York, NY, USA; 1995.

LILIUS, JOHAN.; PALTOR, IVAN P. **vUML: a Tool for Verifying UML Models**, Turku Centre for Computer Science, 1999.

MACHADO, T. A. A. S. da R. **Reengenharia da Interface do Ambiente SEA**. Dissertação de Mestrado, Florianópolis, UFSC. 2007.

MACIEL, P.R.M.; LINS,R.D.; CUNHA, P.R.F. **Introdução às Redes de Petri e Aplicações**, 10a Escola de Computação da Sociedade Brasileira de Computação, Instituto de Computação da UNICAMP, 187pp., Campinas, Julho 1996.

MANOLIOS, PANAGIOTIS; SUBRAMANIAN, GAYATRI; VROON, DARON. **Automating ComponentBased System Assembly**. In: *Proceedings of the international symposium on Software testing and analysis (ISSTA)*. 2007.

MAUDE. Disponível em: <http://maude.cs.uiuc.edu/>. Acesso em: 9 novembro 2010.

MEYER, ERIC.; SOUQUIÈRES, JEANINE. **A systematic approach to transform OMT diagrams to a B specification.** In: *Proceedings of the Formal Method Conference, ser*. LNCS1708. Springer-Verlag, 1999.

MOKHATI, FARID.; GAGNON, PATRICE.; BADRI, MOURAD. **Verifying UML Diagrams with Model Checking: A Rewriting Logic Based Approach**, qsic, pp.356-362, Seventh International Conference on Quality Software (QSIC 2007), 2007.

MOUAKHER, I.; LANOIX, A.; SOUQUIÈRES, J. **Component Adaptation: Specification and Verification.** In: *Proceedings of the International Workshop on Component-Oriented programming, (WCOP)*. 2006

OMG. **OCL**. Specification, versão 2.3.1, OMG, jan.2012.

OMG. **Unified Modeling Language**: Superstructure, versão 1.5, OMG, mar. 2003.

OMG. **Unified Modeling Language**: Superstructure, versão 2.0, OMG, jul. 2005.

OMG. **Unified Modeling Language**: Superstructure, versão 2.4.1, OMG, ago. 2011.

PENDER, T. **UML, a Bíblia**. Rio de Janeiro: Elsevier, 2004.

PIPE. **Platform Independent Petri net Editor 2**, versão 2.5. Disponível em: <<http://pipe2.sourceforge.net/.>> Acesso em: 5 novembro 2011.

PLASIL, F.; VISNOVSKY, S. **Behavior Protocols for Software Components.** In*: IEEE Transactions on Software Engineering , Volume 28 Issue 11* . 2002.

PROMELA**. Process or Protocol Meta Language**. Disponível em: << http://spinroot.com/spin/Man/Quick.html>>. Acesso em: 5 setembro 2011.

RENAUX, E.; LEFEBVRE, E. **Component based method for enterprise application design.** In: *Proceedings of 11th International Workshop on Component Oriented Programming* (WCOP'06), 2006.

ROBBINS, J. E.; REDMILES, D. F. **Cognitive support, UML adherence, and XMI interchange in Argo/UML**, Information and Software Technology, 42(2), 71-149, 25 January 2000.

SALDHANA, J. A.; SHATZ, S. M. **UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis.** In *International Conference on Software Engineering and Knowledge Engineering*. Proc. of the Int. Conf. On Software Eng. and Knowledge Eng. (SEKE), Chicago, 2000.

SARTORI, G. M. S. **Suporte à Geração Semi-automatizada de adaptação para componentes no ambiente SEA.** Dissertação de Mestrado, Florianópolis, UFSC 2005.

SCHÄFER, TIMM.; KNAPP, ALEXANDER.; MERZ, STEPHAN. **Model checking UML state machines and collaborations**, Electronic Notes in Theoretical Computer Science 55 (3) 2001.

SILVA, E. L. D.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. 3ª. ed. Florianópolis: Laboratório de Ensino a Distância da UFSC, 2001. 121 p.

SILVA, R. P. **Como Modelar com UML 2**. Florianópolis: Visual Books, 2009.

SILVA, R. P. **Suporte ao Desenvolvimento e Uso de Frameworks e Componentes.** PhD Thesis. Porto Alegre, UFRGS/II/PPGC, março 2000.

SILVA, R. P. **UML**: **Modelagem orientada a Objetos**. Florianópolis: Visual Books, 2007.

SILVA, R. P.; PRICE, R. T. **Suporte ao desenvolvimento e uso de componentes flexíveis**. In: *Proceedings of XIII Simpósio Brasileiro de Engenharia de Software*. Florianópolis: out. 1999. p.13-28.

SOFTEX. Associação para a Promoção da Excelência do Software Brasileiro. **Perspectivas de desenvolvimento e uso de componentes na indústria brasileira de software e serviços.** Campinas: SOFTEX, 2007.

SOMMERVILLE, I. **Engenharia de Software.** 9.ed. São Paulo:Pearson Prentice Hall, 2011.

SPIN. Disponível em: <http://spinroot.com/spin/whatispin.html>. Acesso em: 8 abril 2011.

SZYPERSKI, C. **Component Technology – What, Where and How?.** In: *Proceedings of the 25th International Conference on Software Engineering* (ICSE.03). 2003

SZYPERSKI, C. **Component Software - Beyond Object-Oriented Programming**. New York, USA: Addison–Wesley, 2002.

SZYPERSKI, C. et al. **Sumary of the first international workshop on component-oriented programming**. In: *Proceedins of the International Workshop on Component-Oriented Programming* (WCOP),1.[S.1.:s.n], 1996.

SZYPERSKI, C. et al. **Sumary of the second international workshop on component-oriented programming**. In*: Proceedings of the*

International Workshop on Component-Oriented Programming, (WCOP),2.[S.1.:s,n], 1997.

TEIXEIRA, N. S; SILVA, R.P. **Compatibility Evaluation of Components Specified in UML.** In*: Proceedings of the V Latin American Workshop on Aspect-Oriented Software Development Advanced Modularization Technique.* (LA-WASP). São Paulo, Brazil. September 2011.

TEIXEIRA, N. S; SILVA, R.P. **Compatibility Evaluation of Components Specified in UML.** In*: Proceedings of the XXX International Conference of the Chilean Computer Science Society,* (SCCC). Curicó, Chile. November 2011.

TEIXEIRA, N. S; SILVA, R.P. **Component-oriented Software Development with UML.** In*: Proceedings of the Sixth International Conference on Software Engineering Advances,* (ICSEA). Bacelona, Spain. October 2011.

THIERRY-MIEG, Y; HILLAH, L-M. **UML behavioral consistency checking using instantiable Petri nets**. In*: Innovations in Systems and Software Engineering*, Volume 4, Number 3, October 2008.

VARGAS, T. C. de S. **Suporte à Edição de UML 2 no Ambiente SEA**. Dissertação de Mestrado, Florianópolis, UFSC. 2008.

WEI, W.; TONG, L.**Component behavior relativity analysis**. In: ACM SIGSOFT Software Engineering Notes Volume 33 Issue 2, New York, NY, USA March 2008.